University of West Bohemia
Faculty of Applied Sciences

Doctoral Thesis

in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in specialization
Computer Science and Engineering

Tomáš Pavelka

Hybrid Methods of Automatic Speech Recognition

Supervisor: Prof. Ing. Václav Matoušek, CSc.
Department of Computer Science and Engineering

Pilsen, 2008

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

Disertační práce

k získání akademického titulu
doktor
v oboru
Informatika a výpočetní technika

Tomáš Pavelka

Hybridní metody rozpoznávání řeči

Školitel: Prof. Ing. Václav Matoušek, CSc.
Katedra informatiky a výpočetní techniky

V Plzni, 2008

Prohlášení

Předkládám tímto k posouzení a obhajobě disertační práci zpracovanou na závěr doktorského studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji tímto, že tuto práci jsem vypracoval samostatně, s použitím odborné literatury a dostupných pramenů uvedených v seznamu, jenž je součástí této práce.

V Plzni dne 1. listopadu 2008 Tomáš Pavelka

Abstract

The thesis explores the possibilities of acoustic modeling by neural networks in automatic speech recognition. Today's most used approach to speech recognition is the application of the theory of hidden Markov models (HMMs). When used for recognition of speech the emission probabilities are commonly modeled by mixtures of Gaussian functions. Research suggests that this task can also be done by neural networks. This is often referred to as the hybrid approach and experiments have shown that it can be advantageous in comparison with Gaussian mixture models (GMMs).

The first phase of our work was to take each type of acoustic model and try to achieve the highest possible recognition accuracy. Experiments were carried out to find how the resulting accuracy changes with the number of trainable parameters (i.e. the number of Gaussians per mixture or the number of neurons in the network) and what changes can be made in the training process.

Our previous research into Gaussian mixture models shows that a significant performance increase can be made by introducing context dependent phonetic units, namely decision tree clustered triphones. In this work we have attempted to do the same with neural networks. The next part of the work was to try to compare both kinds of acoustic models in terms of recognition accuracy and speed. We have found that in order to do so the models have to be tested as an integral part of a recognition system.

Tests done with context independent phonetic units have demonstrated that neural networks use their trainable parameters more efficiently than their GMM counterparts. This leads to higher recognition speeds. However, significant increase in recognition accuracy can be achieved by utilizing context dependent phonetic units where neural networks have been applied only with limited success: We have found that neural networks can perform better than GMMs only if the total number of clustered triphones is kept low (which can be beneficial if there is insufficient amount of training data).

Abstrakt

Práce zkoumá možnosti využití neuronových sítí pro akustické modelování v automatickém rozpoznávání řeči. Nejpoužívanější přístup pro rozpoznávání řeči je aplikace teorie skrytých Markovových modelů. Při rozpoznávání řeči jsou emisní pravděpodobnosti často modelovány směsmi Gaussových funkcí. Výzkum ukazuje, že tento problém může být rovněž řešen neuronovými sítěmi. Toto je často nazýváno hybridním přístupem a experimenty ukazují na výhody v porovnání se směsmi Gaussových funkcí.

V první fázi naší práce jsme se pokusili dosáhnout co možná největší úspěšnosti rozpoznávání u obou typů akustických modelů. Byly provedeny experimenty za účelem zjištění, jak se mění výsledná úspěšnost v závislosti na počtu trénovatelných parametrů (t.j. počtu Gaussových funkcí ve směsi nebo počtu neuronů v síti) a jaké změny mohou být provedeny v trénovacím procesu.

Náš dřívější výzkum v oblasti směsí Gaussových funkcí ukázal, že signifikantní zvýšení úspěšnosti může být dosaženo zavedením kontextově závislých fonetických jednotek, jmenovitě trifónů svázaných decision tree clusteringem. V této práci jsme se pokusili o stejný přístup s neuronovými sítěmi. Další část práce byla porovnání obou typů akustických modelů co do úspěšnosti a rychlosti rozpoznávání. Zjistili jsme, že oba druhy modelů musí být testovány jako integrální součásti rozpoznávacího systému.

Testy provedené s kontextově nezávislými fonetickými jednotkami ukázaly, že neuronové sítě využívají trénovatelných parametrů efektivněji než jejich protějšky založené na směsích Gaussových funkcí, což vede k vyšší rychlosti rozpoznávání. Nicméně, významného zvýšení úspěšnosti rozpoznávání lze dosáhnout využitím kontextově závislých fonetických jednotek, kde neuronové sítě byly aplikovány pouze s částečným úspěchem. Zjistili jsme, že neuronové sítě mohou podávat lepší výsledky než směsi Gaussových funkcí pouze pokud celkový počet trifónů po clusteringu je malý (což může být výhoda, pokud je množství trénovacích dat malé).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For humans speech is the most natural way of communication. Since the invention of a computer a substantial amount of resources has been spent worldwide on research and development of software for automatic speech recognition (ASR). Many applications would benefit from the availability of natural speech input, for example telephone enquiry systems, dictation systems, "hands-busy" applications, programs for visually impaired users, even automatic voice translation into foreign languages.

The performance of all automatic speech recognition systems largely depends on the conditions under which they are tested. The first recognizers performed isolated word recognition: the user said a single word from a limited set and the task for the system was to decide which one of those words it was. Those early recognizers were also tuned to recognize words from a single speaker and under good acoustic conditions.

Generally the more restrictions are placed on the form of the recognized speech the easier the task and the higher the achieved accuracy. But the same restrictions that simplify the recognition for the computer apply to the user who in term has less freedom for communicating in natural speech and thus defy the ultimate purpose of ASR: to simplify the human-computer interaction.

Even though the state-of-the-art speech recognition has advanced significantly in the last decades, all of today's automatic speech recognizers are still domain oriented. Generally, the larger the vocabulary needed to cover the domain the harder the task of automatic speech recognition in that domain.

In automatic speech recognition the knowledge used to infer the spoken word sequence from its acoustic representation comes from two sources: the *acoustic model* which estimated the likelihood of the word sequence based on the acoustic input and the *language model* which represents the prior probability of the word sequence. While the latter may in some simpler

cases be created by hand (e.g. in the form of grammar) the former is almost exclusively stochastic. The task of finding the most likely word sequence based on these sources is in speech recognition referred to as *decoding*.

Stochastic models can learn from data given that sufficient amount of representative data is available. For the development of a speech recognizer that would work in a specific domain it is necessary to have a corpus of utterances from that domain with their respective transcriptions.

We have two corpora in Czech language available: The first one has a very limited vocabulary and consists of voice commands to control a game of chess. It's advantage is that it is an easier task where relatively high recognition accuracy can be reached and the trained recognizer can be utilized for live recognition in an actual chess software. The second one deals with train schedule queries and reservations. It is a more challenging task with larger vocabulary where the models that we design can be thoroughly tested.

The most successful mathematical framework that provides efficient methods for stochastic modeling of speech signal is the theory of *hidden Markov models* (HMMs). In speech recognition a subset of HMMs called continuous density hidden Markov models (CDHMMs) is today's most common approach with Gaussian mixture models (GMMs) used for the estimation of the emission probability for the HMMs.

The works of several authors (e.g. [Teb95] or [BM97]) suggest that there are some deficiencies in the common way the speech signal is modeled and that these can be improved by working with artificial neural networks instead of the more conventional Gaussian mixture models. This is often referred to as the *hybrid approach*.

Neural networks have various advantages that allow them to model the speech signal in a more economical way that can save computational resources during recognition. Among the most notable ones is the ability to discriminate among classes (the standard way of training GMMs by maximum likelihood does not guarantee discrimination).

One area where neural networks are not usually used (with exceptions, see [Bou92] or [Coh92]) is the use of context dependent phonetic units. It is well known that if a model of a speech unit is not only based on the currently recognized speech unit but also on the surrounding ones the word accuracy goes up. Some of our research effort should be concentrated on finding out whether the application of neural networks to acoustic modeling can move in this direction.

1.1 Thesis Goals

Due to our previous success with neural network based approach to acoustic modeling [Pav03] we would like to further explore the possibilities of acoustic modeling by neural networks. The partial goals can be broken down into the following:

1. Train GMM based acoustic models with available speech corpora. Context independent as well as context dependent (e.g. triphones) phonetic units should be tested. These models will be later used for automatic labeling of the training data and will also serve for performance comparison.

2. Design and train neural network based acoustic models. There are various aspects of the training process (such as the error criterion, learning rate, or the choice of activation function) which can have an impact on the final performance. These should be tuned to achieve the highest possible accuracy. A special attention should be given to the possibilities of modeling context dependent units by neural networks.

3. Since the acoustic models are an integral part of a whole recognition system they should be tested as such. For this reason an efficient decoder should be implemented. The fact that the decoding algorithm has parameters which can be tuned and have a significant impact on the overall system's performance should be taken into account. Especially the pruning done during the decoding phase may affect the overall performance of the whole system.

4. Evaluate the performance of both kinds of acoustic models. For this reason a set of representative models should be chosen which will demonstrate how the two methodologies compete in terms of recognition speed and accuracy.

1.2 Thesis Outline

Chapter 2 gives an overview of the theory of hidden Markov models, the field of artificial neural networks is covered in chapter 3 and chapter 4 addresses the issues of employing both in the task of automatic speech recognition. All the aspects of our recognition system and the measurement of its performance that are not related to acoustic modeling or decoding are described in chapter 5. Chapter 6 deals with the design and training of hidden Markov model recognizer with continuous density (GMM) emission probability distribution.

The details of our neural network based acoustic model experiments can be found in chapter 7. The design and implementation aspects of an efficient decoder are described in chapter 8 together with how decoding and acoustic models are related and how the parameters of the decoder need to be tuned in order to achieve the best performance. The results of our experiments and the conclusions that we have reached are summarized in chapter 9.

Chapter 2

Hidden Markov Models

2.1 Basic Concepts

The application of the theory of *hidden Markov models* (HMMs) to the problem of speech recognition has been first introduced in mid 1970s and still remains the most popular and successful approach to the problem. The method provides an elegant mathematical framework which allows data-driven training of both word and sub-word phonetic units.

2.1.1 Elements of an HMM

A hidden Markov model (HMM) is a probabilistic finite state automaton which changes state every discrete time unit t and generates an *observation* o_t with a given probability. An HMM formally consists of the following elements:

1. N, the number of states in the model. Let the states be denoted as $S = \{S_1, S_2, \ldots, S_N\}$, and the state at time t as q_t.

2. The state transition probability distribution $A = \{a_{ij}\}$ where

$$a_{ij} = P[q_{t+1} = S_j \mid q_t = S_i]. \tag{2.1}$$

3. A set of emission probabilities $B = \{b_j(o)\}$ where

$$b_j(o) = P(o|S_j). \tag{2.2}$$

   is the probability distribution describing the likelihood of generating (emitting) each possible observation o while in state S_j.

4. The initial state distribution $\pi = \{\pi_i\}$ where

$$\pi_i = P[q_1 = S_i] \tag{2.3}$$

For a and b to represent probabilities they must meet the following conditions:

$$a_{ij} \geq 0,\ b_j(o) \geq 0,\ \forall i, j, o \tag{2.4}$$

$$\sum_j a_{ij} = 1,\ \forall i \tag{2.5}$$

$$\int_o b_j(o)\, do = 1 \tag{2.6}$$

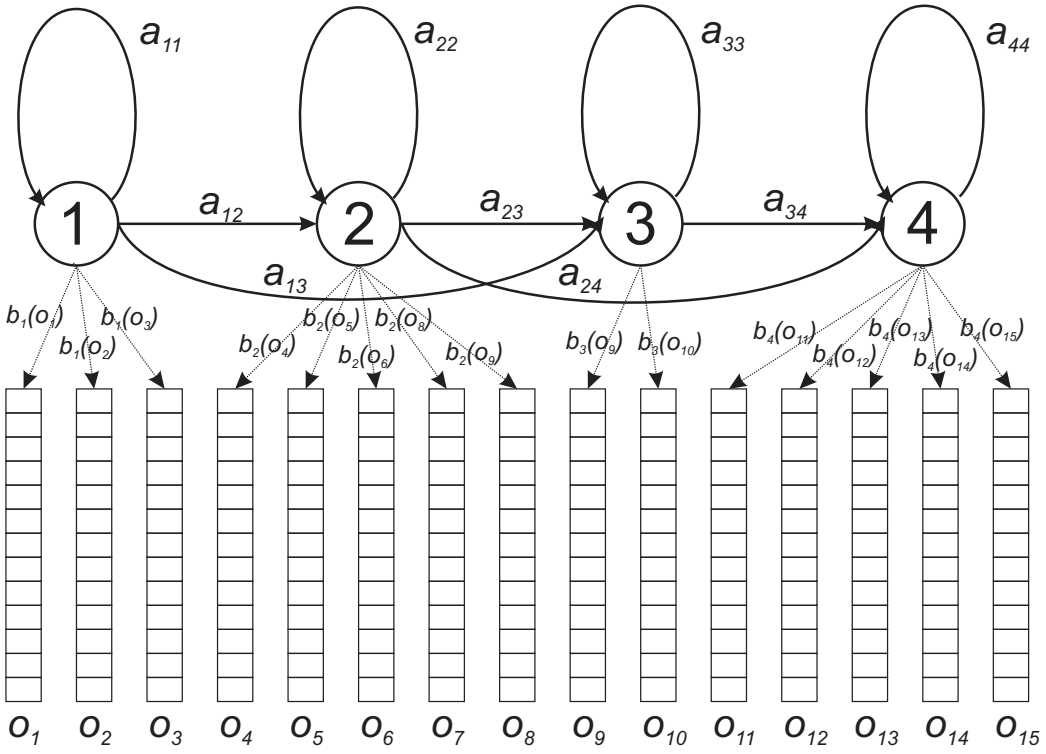

Figure 2.1: A left-right hidden Markov model.

2.1.2 Discrete and Continuous Observation Densities

In many applications the observations may be represented by a finite set of discrete symbols $V = \{v_1, v_2, \ldots, v_M\}$. In that case the element

$$b_j(k) = P(o = v_k | S_j) \tag{2.7}$$

is referred to as discrete observation density. The problem with this approach is that in many cases the observations are continuous signals (or vectors) as it is in automatic speech recognition. Although there exist ways to transform a speech signal into a set of discrete symbols (the technique is called *vector quantization*) it may be beneficial to express the density as a function of the observation vector o.

One of the solutions to this problem is to use some kind of parametric function of which the parameters may be estimated from a set of training data. According to [Rab89] the most general representation of this function for which a reestimation procedure has been formulated is a finite mixture of the form

$$P(o|S_j) = \sum_{m=1}^{M} c_{jm}\mathcal{N}(o, \mu_{jm}, \Sigma_{jm}) \tag{2.8}$$

where c_{jm} is the weight of the m'th mixture component, and $\mathcal{N}(o, \mu, \Sigma)$ can be for example a Gaussian with mean μ and covariance matrix Σ:

$$\mathcal{N}(o, \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} e^{-\frac{1}{2}(o-\mu)'\Sigma^{-1}(o-\mu)} \tag{2.9}$$

where n is the dimensionality of o and $|\Sigma|$ denotes a determinant of the covariance matrix Σ. In order to reduce the computational cost the matrix Σ is in many ASR systems restricted to be diagonal, which requires that the individual components of the observation vector o are statistically independent.

2.1.3 Types of HMMs

Depending on the properties of the state transition matrix a_{ij} different types of HMMs can be distinguished. If any state can be reached from any state in a finite number of steps, we are talking about an *ergodic model*. Although this may be practical in some applications, in speech recognition we would like to model the temporal (time domain) structure of the signal. Hence the state transition coefficients are restricted to have the property

$$a_{ij} = 0, \qquad j < i \tag{2.10}$$

i.e. no transitions are allowed to states whose indices are lower than the current state. Such model is referred to as *left-right model*. In speech recognition it is desired that there is only one entry state to the left-right model i.e.

$$\pi_i = \begin{cases} 0, & i \neq 1 \\ 1, & i = 1 \end{cases} \tag{2.11}$$

In the following pages the left-right model will be assumed and denoted

$$\lambda = (N, A, B) \tag{2.12}$$

2.2 Application of HMMs

Having the HMM defined there are three basic problems of interest to be solved in order for the model to be useful in real-world application, in our case speech recognition. The problems are following:

1. Given the feature vector sequence $O = o_1, o_2, \ldots, o_T$ and a model $\lambda = (N, A, B)$, what is the probability $P(O|\lambda)$, the probability that the model generated the observed sequence of feature vectors? The algorithm used to compute $P(O|\lambda)$ is called *forward procedure*.

2. Given the feature vector sequence O and a model λ, which sequence of states $Q = q_1, q_2, \ldots, q_T$ is optimal in some meaningful sense? The technique for finding these states is the *Viterbi algorithm*.

3. How do we adjust the model parameters $\lambda = (A, B)$ to maximize $P(O|\lambda)$. An iterative procedure for training the model exists: the *Baum-Welch reestimation*.

2.2.1 Solution to Problem 1

The goal is to calculate the probability $P(O|\lambda)$ that a given model λ generated an observation sequence $O = o_1, o_2, \ldots, o_T$. The obvious (but not the wisest as will be shown) approach is to sum the probabilities $P(O, Q|\lambda)$ (i.e. the probabilities of observation sequence O being generated by a state sequence $Q = q_1, q_2, \ldots, q_T$) over all possible state sequences Q.

The probability $P(O, Q|\lambda)$ is the joint probability of O and Q occurring simultaneously given the model λ and can be computed as

$$P(O, Q|\lambda) = P(O|Q, \lambda)P(Q|\lambda) \tag{2.13}$$

where

$$P(O|Q, \lambda) = \prod_{t=1}^{T} P(O_t|Q_t, \lambda) = b_{q_1}(o_1) \cdot b_{q_2}(o_2) \cdot \ldots \cdot b_{q_T}(o_T) \tag{2.14}$$

and

$$P(Q|\lambda) = a_{q_1 q_2} \cdot a_{q_2 q_3} \cdot \ldots \cdot a_{q_{T-1} q_T}. \tag{2.15}$$

By those equations the computing of a single $P(O, Q|\lambda)$ involves $2T - 1$ multiplications. The naive way of computing the probability of O being generated by the model λ is

$$P(O|\lambda) = \sum_{\forall Q} P(O, Q|\lambda) \tag{2.16}$$

Since there are total N^T possible state sequences Q the computation of $P(O|\lambda)$ would require $(2T - 1)N^T$ multiplications and $N^T - 1$ additions.

Fortunately the problem can be solved by dynamic programing. The algorithm is called the *forward procedure*. Let the forward variable $\alpha_t(i)$ be defined as

$$\alpha_t(i) = P(o_1\ o_2\ \ldots\ o_t, q_t = S_i|\lambda) \tag{2.17}$$

meaning the probability of the observation sequence $o_1\ o_2\ \ldots\ o_t$ being generated by the model λ while the last observation o_t was generated by the state S_i. The probability of the state in time $t + 1$ is only dependent on the state the model is in time t and the probability of the model generating the observation O_{t+1} is only dependent on the state the model is in time $t + 1$. That means that the forward variable α_{t+1} can be directly computed from α_t leading to a recursive procedure feasible in terms of computational costs:

1. Initialization:
$$\alpha_1(i) = \pi_i b_i(o_1), \qquad 1 \leq i \leq N \tag{2.18}$$

2. Induction:
$$\alpha_{t+1}(i) = \left[\sum_{j=1}^{N} \alpha_t(i)a_{ij}\right] b_i(o_{t+1}), \qquad \begin{array}{l} 1 \leq t \leq T - 1 \\ 1 \leq j \leq N \end{array} \tag{2.19}$$

3. Termination:
$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{2.20}$$

2.2.2 Solution to Problem 2

The second problem has not a unique solution because there can be several criteria for deciding an optimal state sequence that generated the given observation sequence. For example the individual most likely states at each time t may be considered. The flaw of this criterion is that it does not consider whether the optimal sequence is valid in terms of allowed state transitions. According to this criterion two subsequent states $q_t = S_i$ and $q_{t+1} = S_j$ may

be found individually most likely even if the transition probability a_{ij} is zero. If this criterion was applied to the task of speech recognition it could lead to recognition sequences of phonemes which do not form valid words.

The most widely used criterion which overcomes the previously described difficulty is one that seeks the **single most likely state sequence**, i.e. maximizes $P(Q|O,\lambda)$. The solution is similar to the forward procedure and is called the *Viterbi algorithm*. Let $\delta_t(i)$ be the probability of getting to state S_i at time t through the most likely path:

$$\delta_t(i) = \max_{q_1,q_2,\ldots,q_{t-1}} P[q_1\ q_2\ \ldots\ q_t = i, o_1\ o_2\ \ldots\ o_t|\lambda] \tag{2.21}$$

Usually when this algorithm is applied the optimal state sequence Q^* is more important than its probability $P(Q^*|O,\lambda)$. In order to retrieve this sequence a backpointer $\psi_t(i)$ is stored for each value of $\delta_t(i)$ pointing to the state in time $t-1$ from which the optimal state in time t was reached. The complete algorithm is:

1. Initialization:

$$\begin{aligned} \delta_1(1) &= \pi_i b_i(o_i), \qquad 1 \le i \le N && (2.22)\\ \psi_1(1) &= 0. && (2.23) \end{aligned}$$

2. Induction:

$$\delta_j(t) = \max_{1\le i\le N}[\delta_i(t-1)a_{ij}]b_j(o_t) \qquad \begin{matrix} 2 \le t \le T \\ 1 \le j \le N \end{matrix} \tag{2.24}$$

$$\psi_j(t) = \operatorname*{argmax}_{1\le i\le N}[\delta_i(t-1)a_{ij}] \qquad \begin{matrix} 2 \le t \le T \\ 1 \le j \le N \end{matrix} \tag{2.25}$$

3. Termination:

$$\begin{aligned} P(Q^*|O,\lambda) &= \max_{1\le i\le N}[\delta_i(T)] && (2.26)\\ q_T^* &= \operatorname*{argmax}_{1\le i\le N}[\delta_i(T)] && (2.27) \end{aligned}$$

4. Most likely state sequence backtracking:

$$q_t^* = \psi_{q_{t+1}^*}(t+1) \qquad T-1 \ge t \ge 1 \tag{2.28}$$

2.2.3 Solution to Problem 3

The most difficult problem is to find the model parameters (A, B, π) that would maximize the probability that the model generated an observation sequence O. According to [Rab89] there exists neither an analytical solution nor a numerical method that would find the model parameters which globally maximize the probability $P(O|\lambda)$. There are several methods that iteratively adjust the model parameters in order to achieve a local maximum of $P(O|\lambda)$. The one described here is called *Baum-Welch re-estimation*.

Let's first define a *backward variable* $\beta_t(i)$, the probability of partial observation sequence from time $t+1$ to end starting in the state $q_t = S_i$ given the model λ:

$$\beta_t(i) = P(o_{t+1}\ o_{t+2}\ \ldots\ o_T|q_t = S_i, \lambda) \tag{2.29}$$

The backward variable can be computed in a similar fashion as the forward variable:

1. Initialization:
$$\beta_T(i) = 1, \qquad 1 \leq i \leq N \tag{2.30}$$

2. Induction:
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), \qquad \begin{array}{l} T \geq t \geq T-1 \\ 1 \leq j \leq N \end{array} \tag{2.31}$$

Let $\gamma_t(i)$ be the probability of being in state S_i at time t:

$$\gamma_t(i) = P(q_t = S_i|O, \lambda) = \frac{P(O, q_t = S_i|\lambda)}{P(O|\lambda)} \tag{2.32}$$

Note that

$$\begin{aligned} P(O, q_t = S_i|\lambda) &= P(o_1\ \ldots\ o_t, q_t = S_i|\lambda) P(o_{t+1}\ \ldots\ o_T|q_t = S_i, \lambda) = \\ &= \alpha_t(i)\beta_t(i) \end{aligned} \tag{2.33}$$

which means that $\gamma_t(i)$ can be computed as

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} \tag{2.34}$$

Let's further define the probability of transiting from state S_i at time t to state S_j at time $t+1$ as

$$\xi_t(i, j) = P(q_t = S_i, q_{t+1} = S_j|O, \lambda) \tag{2.35}$$

Again the probability can be rewritten in terms of forward and backward variables:

$$\begin{aligned} \xi_t(i,j) &= \frac{P(q_t = S_i, q_{t+1} = S_j, O|\lambda)}{P(O|\lambda)} = \\ &= \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} \end{aligned} \tag{2.36}$$

Note that the term $P(O|\lambda)$ is the solution to problem 1 (see equation 2.20)

If the probability $\gamma_t(i)$ is summed over the time t the result can be interpreted as the expected number of times that state S_i is visited during the generating of the observation sequence O. If time $t = T$ is excluded

$$c(i) = \sum_{t=1}^{T-1} \gamma_t(i), \tag{2.37}$$

the sum can be interpreted as the expected number of transitions made from state S_i. Similarly the sum of the probabilities $\xi_t(i,j)$ over the time t

$$c(i,j) = \sum_{t=1}^{T-1} \xi_t(i,j) \tag{2.38}$$

leads to the expected number of transitions from state S_i to state S_j.

Having these variables defined the formulas for new estimations of model parameters can be derived. The new estimate for the initial state distribution can be computed as

$$\pi_i^* = \gamma_1(i) \tag{2.39}$$

i.e. the probability of being in state S_i at time $t = 1$. The new estimate of state transition probability is

$$a_{ij}^* = \frac{c(i,j)}{c(i)} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{2.40}$$

i.e. the expected number of transitions from S_i to S_j divided by the expected number of visits of S_i. The formula for the new value of the state emission probability depends on whether the probability density is discrete or continuous (see section 2.1.2). For the case of discrete density the new parameter $b_j^*(k)$ is calculated as

$$b_j^*(k) = \frac{\sum_{\substack{t=1 \\ o_t = v_k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.41}$$

i.e. the probabilities $\gamma_t(j)$ of being in state S_i are summed but only for the cases where the desired symbol v_k is observed. Those are then divided by the expected number of times the state S_j is visited.

The formulas for estimating the parameters of continuous density functions will be given only for the case of a single Gaussian. For the more complicated cases see [Bil98], [Rab89] or [You02]. The new estimate for the mean of the Gaussian density is calculated as a weighted average of the observation sequence with the variable $\gamma_t(j)$ taken as weight:

$$\mu_j^* = \frac{\sum_{t=1}^{T} \gamma_t(j) o_t}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.42}$$

The new values for the covariance matrix Σ_j are computed in a similar fashion:

$$\Sigma_j^* = \frac{\sum_{t=1}^{T} \gamma_t(j)(o_t - \mu_j)(o_t - \mu_j)'}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.43}$$

It has been proven (see [Bil98] and [Rab89] for further references) that if the current model parameters $\lambda = (A, B, \pi)$ are used to re-estimate the new parameters $\lambda^* = (A^*, B^*, \pi^*)$ then either

$$P(O|\lambda^*) = P(O|\lambda) \tag{2.44}$$

and λ locally maximizes the probability function or

$$P(O|\lambda^*) > P(O|\lambda), \tag{2.45}$$

i.e the new model λ^* is more likely to have generated the observation sequence O than the current one. The Baum-Welch re-estimation is a special case of the Expectation-Maximization (EM) algorithm of statistics. For details on this algorithm consult [Bil98]

2.3 Training of HMM systems

The previous section described how model parameters can be adjusted in order to achieve a higher probability that a model generated a given observation sequence. The process of iteratively refining the model parameters until the change in the probability $P(O|\lambda)$ is sufficiently small is referred to as *training*. This section will address some of the problems faced when training HMMs for speech applications.

2.3.1 Multiple Observation Sequences

In the application of HMMs for speech recognition the left-right model is used (see section 2.1.3). The problem with this topology is that a single observation sequence is not sufficient for the estimation of the model parameters. This is because there are usually only few subsequent observations belonging to one state and after this state is left it is not possible to return because of the restriction imposed by equation 2.10.

Multiple observation sequences are necessary to provide for reliable estimates of the model parameters. The re-estimation procedure needs to be changed to compute with a set of observation sequences.

$$O = [O^{(1)}, O^{(2)}, \ldots, O^{(R)}] \tag{2.46}$$

where $O^{(r)} = o_1^{(r)}, o_2^{(r)}, \ldots, o_{T_r}^{(r)}$ is the rth observation sequence. The probability function being maximized in this case is

$$P(O|\lambda) = \prod_{r=1}^{R} P(O^{(r)}|\lambda) \tag{2.47}$$

$$= \prod_{r=1}^{K} P_r \tag{2.48}$$

The new re-estimation formulas are obtained by summing the terms in the numerator and denominator in the original formulas over all r observation sequences. The state transition probability is

$$\begin{aligned} a_{ij}^* &= \frac{\sum_{r=1}^{R} \sum_{t=1}^{T_r-1} \xi_t^r(i,j)}{\sum_{r=1}^{R} \sum_{t=1}^{T_r-1} \gamma_t^r(i)} \\ &= \frac{\sum_{r=1}^{R} \frac{1}{P_r} \sum_{t=1}^{T_r-1} \alpha_t^r(i) a_{ij} b_j(o_{t+1}^{(r)}) \beta_t^r(j)}{\sum_{r=1}^{R} \frac{1}{P_r} \sum_{t=1}^{T_r-1} \alpha_t^r(i) \beta_t^r(i)} \end{aligned} \tag{2.49}$$

The state emission probability (for the case of discrete density) is

$$\begin{aligned} b_i(k)^* &= \frac{\sum_{r=1}^{R} \sum_{\substack{t=1 \\ o_t=v_k}}^{T_r} \gamma_t^r(j)}{\sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t^r(j)} \\ &= \frac{\sum_{r=1}^{R} \frac{1}{P_r} \sum_{\substack{t=1 \\ o_t=v_k}}^{T_r} \alpha_t^r(i) \beta_t^r(i)}{\sum_{r=1}^{R} \frac{1}{P_r} \sum_{t=1}^{T} \alpha_t^r(i) \beta_t^r(i)} \end{aligned} \tag{2.50}$$

The formulas for estimating the parameters of continuous density functions can be found in [You02]. The initial state distribution needs not to be estimated for the left right model (see equation 2.11).

2.3.2 Embedded Training

The previously discussed strategies require that if there are more HMMs in the system, the training data for those HMMs need to be separated, i.e. each of those models has to have assigned its own observation sequence set. *Embedded training* is useful when there are data consisting of observation sequences that are expected to have been generated by a number of concatenated HMMs.

This is the case when continuous speech needs to be processed. The speech signal is modeled as a sequence of phonetic units[1] and each phonetic unit has a corresponding hidden Markov model. When those HMMs are concatenated they may form words and sentences.

The difficulty of this approach comes from the transition probabilities between the phonetic units since these cannot be easily estimated. One solution is to use *non emitting states* [You02] as the entry and exit states of the HMMs representing phonetic units. The idea is that those states have transition probabilities but generate no observations. When two HMMs are concatenated the first one shares its non-emitting exit state with the non-emitting entry state of the second model. The formulas from problems 1-3 need to be changed to allow transitions from and to non-emitting states in time $t + \delta t$ resp. $t - \delta t$, i.e. between the discrete time units. For details see [You02].

Unfortunately the boundaries of phonetic units cannot be easily found but are necessary if the training technique described in the previous section is to be used. It is possible to segment the speech signal manually but it is rather tedious and time consuming task. It is possible to estimate the boundary locations by forced Viterbi alignment (see section 4.5) but a set of already trained models must be available for this task.

In embedded training a composite HMM is constructed for each training utterance by concatenating individual HMMs of phonetic units according to the phonetic transcription of the utterance. The forward-backward algorithm is then applied to all training utterances and the sums (see eq. 2.49 and 2.50) needed for new estimates of transition and emission probability densities are accumulated. The embedded re-estimation is usually repeated several times, until the performance of the recognition system stops to increase.

[1] In a simple case a phonetic unit is roughly equal to phoneme.

2.4 Limitations of HMMs

Although hidden Markov models found their usage in almost every state-of-the-art recognition system, there are several well-known weaknesses to mention:

- The First-Order Assumption: All probabilities depend only on the current state. While this greatly reduces computational load, it is not true for speech signal, acoustic distributions are affected by recent history. The consequence is that HMMs have difficulty modeling coarticulation.

- The Independence Assumption: Successive frames of speech are independent. As a result the probability of a sequence of observations can be computed as a product of probabilities of individual feature vector observations, i.e.

$$P(o_1, o_2, \ldots, o_T) = \prod_{i=1}^{T} P(o_i).$$

  Again this is not true for speech signal.

- Standard probability density models used in HMMs (discrete, continuous, see section 2.1.2) do not accurately model the true density of acoustic space.

- During Baum-Welch re-estimation the standard training criterion (Maximum Likelihood) does not guarantee discrimination among acoustic models. Maximum Mutual Information (see [Rab89]) training criterion can be used to solve this problem, but it is very complex and difficult to implement properly.

Chapter 3

Artificial Neural Networks

3.1 Fundamentals

Artificial neural networks (ANNs) are parallel computational models originally inspired by studies in biological neural structures. The most valued feature of an ANN is the possibility to adapt its parameters (called *weights*) from a set of learning examples. This is a useful feature when the problem is not well understood but training data is available.

The problems to which the theory of artificial neural networks has been successfully applied include pattern classification, speech recognition and understanding, function approximation, image compression and restoration, forecasting and time series prediction, system control and many others.

While the first neural models were designed as models of their biological counterparts, the today's most widely utilized artificial neural networks are more closely related to traditional mathematical and statistical models such as statistical regression models or non parametric pattern classifiers.

The first artificial neurons were introduced by McCulloch and Pitts [MP43] and were intended as simplified models of biological neurons and a demonstration of the ability to perform computational tasks. The so called *connectionist* approach was then largely criticized mainly for its inability to learn to solve some elementary mathematical problems, mainly the computation of the XOR function. The interest in neural networks was renewed by the introduction of the backpropagation algorithm [Rum86], which allowed for training of layered networks.

The overview of all architectures of neural networks is beyond the scope of this thesis and can be found, for example in [Has95] or [Sar02]. This chapter will be aimed at the layered networks and their modifications which were previously successfully applied in the field of automatic speech recognition.

3.1.1 Model of a Neuron

Given the activations of units in previous layer y_j and weights w_{jk} the total input of unit k is s_k, computed as

$$s_k = \sum_j w_{jk} y_j + \theta_k \tag{3.1}$$

where θ_k is called a *bias*. The total input is then transformed using *activation function* $F(s_k)$ to give activation y_k

$$y_k = F(s_k). \tag{3.2}$$

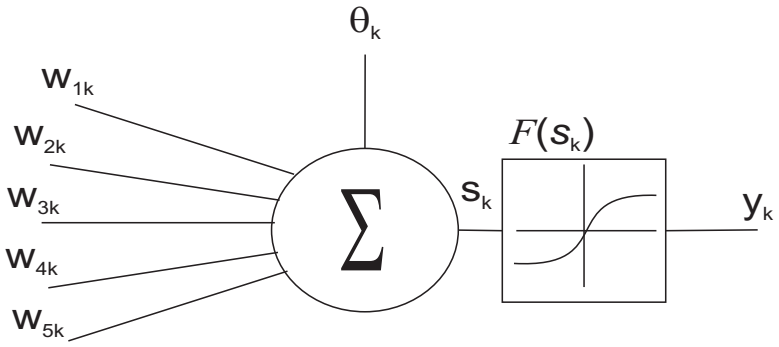

Figure 3.1: Model of a neuron

Function	**Formula**
Identity	$y_k = s_k$
Logistic	$y_k = \frac{1}{1+e^{-s_k}}$
Symmetric Logistic	$y_k = \frac{2}{1+e^{-s_k}} - 1$
Hyperbolic Tangent	$y_k = \frac{e^{2s_k}-1}{e^{2s_k}+1}$
Softmax	$y_k = \frac{e^{s_i}}{\sum_j e^{s_j}}, \quad \forall j$ in the layer

Table 3.1: Common activation functions used in multi-layer perceptrons

Let's now explain the importance of the bias term θ. By setting $s_k = 0$ equation (3.1) rewritten as

$$w_{1k}y_1 + w_{2k}y_2 + \ldots + w_{Nk}y_N + \theta_k = 0 \tag{3.3}$$

draws a hyperplane dividing the N-dimensional space of y_j into two parts, in case of threshold activation function producing "on" output on one side

and "off" output on the other. In the case of classification problem one side is class "A" and the other class "B". If there is no bias, i.e. $\theta_k = 0$ the hyperplane passes through the origin of the space. If the inputs belonging to class "A" are distributed around the origin, class "A" cannot be correctly recognized.

For easier implementation the bias can be used as a weight from neuron giving constant activation $y_{N+1} = 1$:

$$\theta_k = w_{N+1,k} y_{N+1}. \tag{3.4}$$

This modification allows the use of the same formulas for training both weights and biases.

3.1.2 Multi-layer Perceptrons

Artificial neural networks consisting of neuron models described in section 3.1.1 organized in layers are referred to as *multi-layer perceptrons*. Adjacent layers are fully interconnected, i.e. each neuron is connected with all neurons in the previous and in the following layer. In the literature there is not a complete agreement on what should be considered a layer when counting their number. As opposed to some sources which count only the layers with weights (e.g. [Has95]) in this thesis the name three layer perceptron refers to a network with three layers of neurons and two layers of weights. Those three layers will further be called *input, hidden and output*.

The introduction of the backpropagation algorithm allowed the application of layered neural networks to many complex problems. An important question is how many hidden layers should the network have in order to successfully approximate the correct mapping from its inputs to its outputs. It has been empirically confirmed by [HL87] that the addition of a second hidden layer does not speed up the convergence of backpropagation algorithm when training a classifier in two dimensions, even if the class boundaries are complex. A mathematical proof was later given by [Cyb89] that a three layer perceptron with logistic activation functions (see Table 3.1) can approximate to any desired precision any continuous real valued function, given that the hidden layer has a sufficient number of neurons.

Another useful feature of multilayer perceptron is that if it is trained as a 1-of-N classifier using mean squared error or similar criterion, then its output activations will approximate the class posterior probability $P(class|input)$ with an accuracy that improves with the size of the training set (see [BM97] for the proof and further references). This fact has also been experimentally confirmed, see [Pav03] and [Teb95].

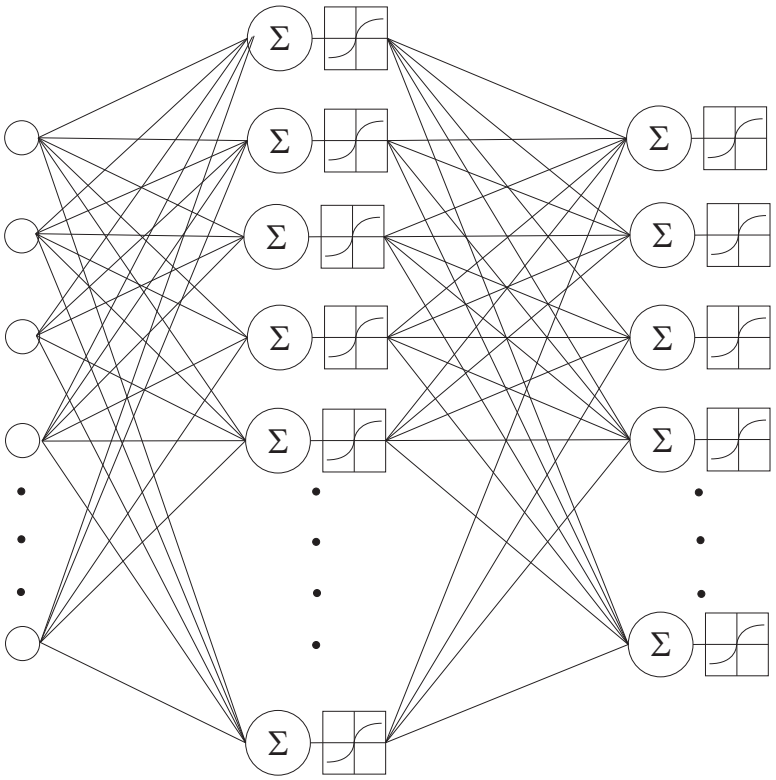

Figure 3.2: Multi-layer perceptron with identity activation function in the input layer.

3.1.3 Recurrent Neural Networks

It has been stated in the previous chapter that a layered neural network can be trained to approximate any existing function. Suppose that the input vectors of the network are part of a time series and further information required for the task (i.e. classification) may be acquired from the development of the input vectors in time. In the case of a multi-layer perceptron it is possible to introduce a *context window*: when processing the input vector x_t belonging to time instance t, several adjacent input vectors are added to create a new input vector:

$$X_{t-c}^{t+d} = \{x_{t-c}, \ldots, x_t, \ldots, x_{t+d}\}. \tag{3.5}$$

While this have been proven to be very effective in the case of speech recognition [BM97], [Teb95], [Pav03] it also has disadvantages:

- The optimal size of the context window is unknown prior to training and must be determined empirically.

- The computational requirements rise with the size of the context window, even though subsequent input vectors with their respective contexts differ only very little.

If recurrent weights are added to the network its approximating power is not extended, but fewer trainable parameters (weights) may be required for the task. Only one version of such network will be discussed since the theory of dynamics of recurrent networks (RNNs) is beyond the scope of this thesis (see [Has95], [KS96], [Rob89], [Sar02] for more general information on RNNs).

Figure 3.3 shows a network used by [Rob94] to map a sequence of speech frames to a sequence of their corresponding phonetic labels. The network is fed by the current input vector $u(t)$ along with the current *state vector* $x(t)$. These are passed through a standard feed forward network to produce the output vector $y(t)$ and the next state vector $x(t+1)$.

The training algorithm for such network is called backpropagation through time an may be found in [Rob94].

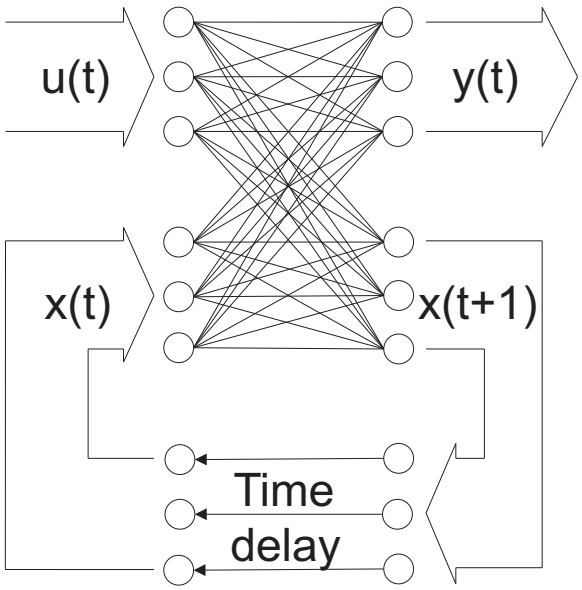

Figure 3.3: Recurrent neural network for speech recognition (from [Rob94])

3.2 Training

3.2.1 Backpropagation

The *backpropagation* algorithm or the *generalized delta rule* is the most widely used supervised learning algorithm an thus will be described in more detail. In supervised learning the target outputs are available for each set of inputs. First, a *criterion function* that measures the degree of approximation must be defined. A common function is the SSE (see below) but it can be beneficial to use other criterion functions as will be discussed later. The learning algorithm seeks to adjust the weights in order to minimize the criterion function. If this function is differentiable, *gradient descent learning* can be used to locally minimize the error of the network.

In gradient descent learning the weight change $\triangle w_{jk}$ is chosen with respect to the error gradient $\frac{\partial E}{\partial w_{jk}}$. This is repeated until the error is sufficiently small. The notation has been taken from [KS96].

Let the set of network inputs p be called *input pattern* and the set of desired network outputs d^p be called *output pattern*. The *summed squared error* (SSE) is computed as:

$$E^p = \frac{1}{2}\sum_{o=1}^{N_o}(d_o^p - y_o^p)^2, \tag{3.6}$$

where o is index of an output unit. Using chain rule, we can write

$$\frac{\partial E^p}{\partial w_{jk}} = \frac{\partial E^p}{\partial s_k^p}\frac{\partial s_k^p}{\partial w_{jk}}. \tag{3.7}$$

By equation (3.1) we see that

$$\frac{\partial s_k^p}{\partial w_{jk}} = y_j^p. \tag{3.8}$$

We define the update rule

$$\delta_k^p = -\frac{\partial E^p}{\partial s_k^p}. \tag{3.9}$$

To compute δ_k^p we apply the chain rule

$$\delta_k^p = -\frac{\partial E^p}{\partial s_k^p} = -\frac{\partial E^p}{\partial y_k^p}\frac{\partial y_k^p}{\partial s_k^p}. \tag{3.10}$$

By equation (3.2) we see that

$$\frac{\partial y_k^p}{\partial s_k^p} = F'(s_k^p), \tag{3.11}$$

which is the derivative of the activation function of the unit k. Now let us consider two cases. First assume that the unit k is an output unit $k = o$. Then, from the definition of E^p

$$\frac{\partial E^p}{\partial y_o^p} = -(d_o^p - y_o^p), \tag{3.12}$$

and by equations (3.10) and (3.11)

$$\delta_o^p = -(d_o^p - y_o^p)F'(s_k^p). \tag{3.13}$$

Second, assume that the unit k is a unit in the hidden layer $k = h$,

$$\begin{aligned} \frac{\partial E^p}{\partial y_h^p} &= \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial s_o^p}\frac{\partial s_o^p}{\partial y_h^p} = \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial s_o^p}\frac{\partial}{\partial y_h^p}\sum_{j=1}^{N_h}\omega_{jo}y_j^p = \\ &= \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial s_o^p}\omega_{ho} = -\sum_{o=1}^{N_o}\delta_o^p w_{ho}. \end{aligned} \tag{3.14}$$

Now we can compute

$$\delta_h^p = F'(s_h^p)\sum_{o=1}^{N_o}\delta_o^p w_{ho}. \tag{3.15}$$

At this point we have a way to compute the error gradient

$$\frac{\partial E^p}{\partial w_{jk}} = -\delta_k^p y_j^p. \tag{3.16}$$

3.2.2 Alternative Error Criterion

A commonly used alternative to summed squared error criterion is the *cross entropy* (CE) (or *relative entropy*). According to [Has95] it is well suited for training the net as a class probability estimator. Empirical observations show that the usage of cross entropy error criterion in training multi layer perceptrons for speech recognition speeds up the training by reducing the number of training cycles [BM97].

$$E^p = -\sum_{o=1}^{N_o}(d_o^p \log y_o^p) + (1 - d_o^p)\log(1 - y_o^p). \tag{3.17}$$

The output layer delta (see equation 3.13) is then computed as

$$\delta_o^p = \frac{\partial E^p}{\partial y_o^p}F'(s_k^p) = \left(\frac{1 - d_o^p}{y_o^p - 1} + \frac{d_o^p}{y_o^p}\right)F'(s_k^p). \tag{3.18}$$

3.2.3 Weight Update Strategies

There exist two general types of weight update strategies: incremental and batch. In *incremental* training the weights are updated after each learning pattern is processed. In batch training the error gradients are summed for all the training patterns and the weights are updated with respect to this global error gradient. The batch version has a much slower convergence, but always leads towards a local minimum of the error function [Sar02]. In case of the incremental version the order in which the training patterns are presented may slower or prevent the convergence, hence it is advised to randomly shuffle the training data before incremental training.

The standard backpropagation algorithm [Rum86] used a simple update scheme where the change in weight is the error gradient multiplied by a *learning rate* η:

$$\triangle w_{jk} = -\eta \frac{\partial E^p}{\partial w_{jk}}. \tag{3.19}$$

The choice of an optimal learning rate is rather intricate since too large learning rates may lead to oscillations around the error minimum [KS96] and too small learning rates cause slow convergence. A possible solution for avoiding the oscillations is to introduce a *momentum term* and base the update on its value in the previous step:

$$\triangle w_{jk}(t) = -\eta \frac{\partial E^p}{\partial w_{jk}}(t) + \mu \triangle w_{jk}(t-1). \tag{3.20}$$

The momentum term μ scales the influence of the previous update step on the current. It should render the training more stable and increase convergence in shallow regions of the error function.

There are many other update strategies of which the *resilient propagation* (RPROP) [Rie93] can serve as example of the more sophisticated ones. It tries to eliminate the need for setting the learning rate by having a separate update step for each weight, which is adapted throughout the learning process. Another of its advantages is, that it does not consider the magnitude of the error gradient, which can often be misleading, only its sign. An example of its usage in speech recognition can be found in [Pav03].

3.3 Generalization

One of the major advantages of neural networks is their ability to generalize, i.e. to produce accurate output for an input pattern it has not been trained with. Usually the reason for using neural networks is that for some inputs

we know what the outputs should look like, and we cannot find a suitable function that maps them.

Unfortunately generalization is not always possible, according to [Sar02] there are three necessary—although not sufficient—conditions to generalization:

1. A function relating correct outputs to inputs must exist, i.e. the inputs must carry sufficient information.

2. This function must be, in some sense, smooth. A small change in the inputs should produce small change in the outputs.

3. The training cases must be sufficiently large and representative subset of all the cases we wish to generalize.

In practice the better the problem satisfies those conditions, the more accurate the generalization is. When the network is trained on noisy data (such is the case with speech recognition) an unwanted phenomenon called *overfitting* may occur. During the initial phases of training the weights are adjusted to fit major features in the data. As the training proceeds the weights are updated to further reduce the error on the training set and thus fitting the noise in the data. This may lead to a decrease of performance on data not used during training. The technique for solving this problem is called *cross validation*. Usually the training data consists of three separate parts:

- The **training set**, of which the error is minimized during training.

- A small **validation set** used to track approximation of the generalization error during training. When the error on the validation set starts to rise, the training should cease.

- A **test set** to finally check the overall performance of the network. The data from the validation set should not be part of the test set.

Chapter 4

Automatic Speech Recognition

The ultimate goal of *automatic speech recognition* (ASR) is to derive a sequence of words from acoustic signal. A more general task would be automatic speech understanding where the resulting word sequence is further processed to extract meanings.

The first speech recognition systems developed in 1950 were based on *template matching* this technique compares templates (which are often just actual examples of speech signal) with unknown words. This approach lacks robustness and is only suitable for the recognition of isolated words since it is not feasible to have a template of every possible sentence.

Another attempt at the problem was to utilize expert knowledge (from linguists, phoneticians, spectrogram readers and others), and has met with only a limited success[1]. Nonetheless a sort of expert knowledge enhancement of a ANN/HMM hybrid recognizer was recently successfully used by Hossom [Hos00] in the task of automatic alignment of phoneme labels.

Today's most successful speech recognition systems employ probabilistic models of sub-word units. This chapter will discuss the issues faced when designing, implementing and training such system with the emphasis on so called *hybrid approach* which combines the advantages of hidden Markov models and artificial neural networks.

4.1 Components of an ASR system

The components of a usual speech recognition system along with a graphical representation of their outputs are shown in Figure 4.1. Each individual

[1] "Every time I fire a linguist, the performance of our speech recognition system goes up." F. Jelinek, Workshop on Evaluation of NLP Systems, Wayne, Pennsylvania, US, December 1988

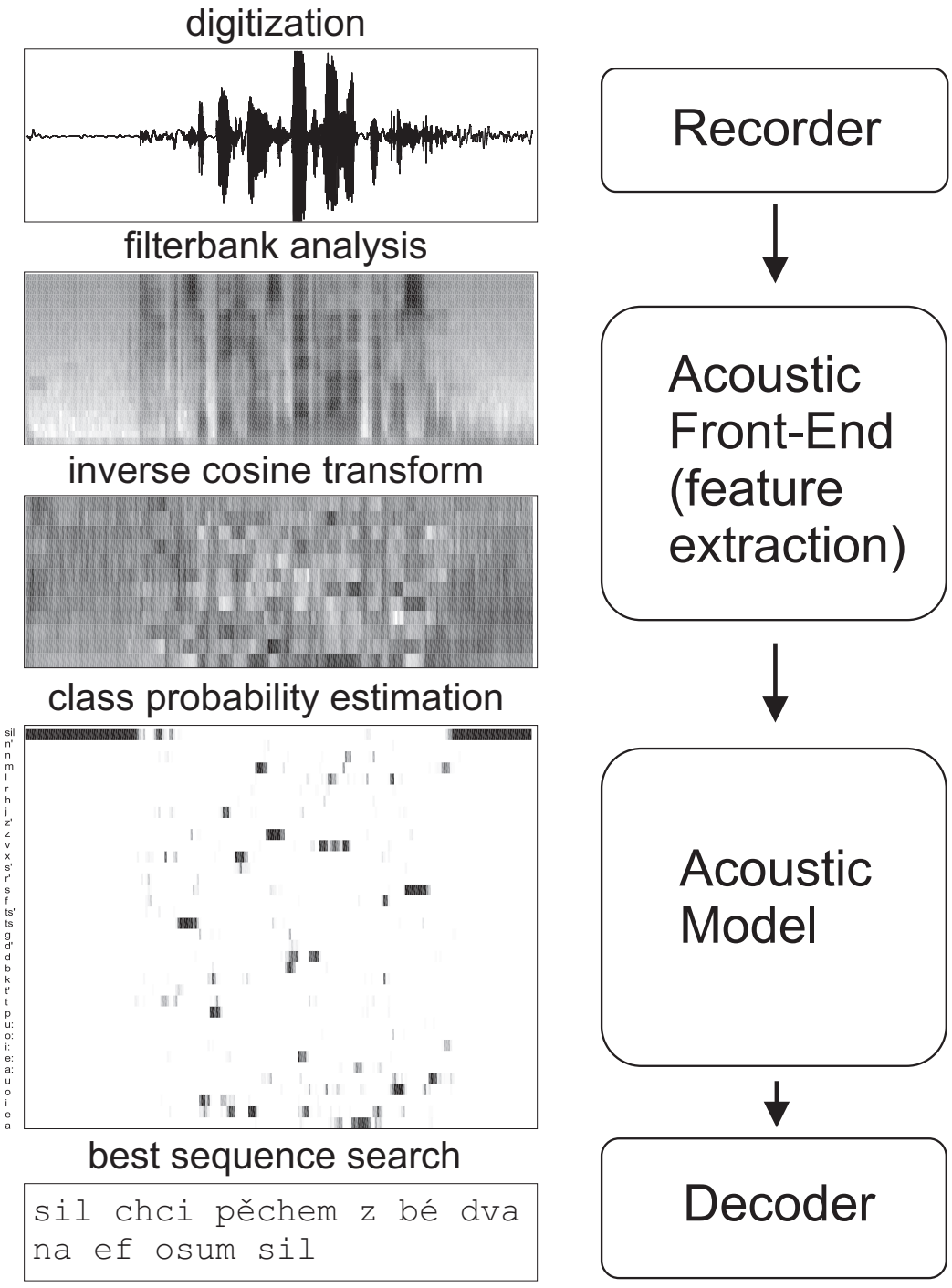

Figure 4.1: Components of an automatic speech recognition system. Mel frequency filterbank was used during feature extraction. The class probabilities were generated by a multi-layer perceptron.

component can be a stand alone module allowing for separate design and implementation.

- **Recorder**. The recognition system is usually designed and trained to deal with speech only. The key issue during the recording phase is to determine the beginning and end of the utterance. This bring the need for advanced silence and background noise detection methods (see [EP04b]). The recorder component as discussed here is responsible for the control of the recording, the actual recording and digitization is done by the sound card and microphone. The most common sampling rate for speech applications is 16KHz with 16 bits stored per sample.

- **Audio Front End**. There are two main tasks for feature extraction: the reduction of data size, which decreases the computational load on the following components, and the extraction of such features that would allow good separation of the recognized classes of speech sounds during acoustic modeling. Prior to feature extraction the signal is split into small (tens of milliseconds) segments, often called *frames*. From each frame a feature vector is computed.

- **Acoustic Model**. The acoustic model provides a local match, i.e. estimates to which phonetic class does a speech frame belong to. The acoustic model does not produce a single hard choice but rather assigns scores or probabilities to all phonetic classes being modeled. The emission probability component of an HMM (see section 2.1.2) is an example of an acoustic model. Later it will be shown that it can be advantageous to employ a neural network for this purpose.

- **Decoder**. The decoder searches the local matches in order to obtain a global math, i.e. the best scoring sequence of words. Usually some restrictions are imposed during the search, for example the set (dictionary) of valid words is limited or there may be limits on the way individual words can follow one another (syntactic limitations). Generally any such restriction increases the recognition accuracy given that the utterance being recognized obeys those restrictions. The Viterbi algorithm (see section 2.2.2) is an example of such search.

4.2 Feature Extraction

Before feature extraction the raw digitized speech signal is segmented into small *frames* usually 10-50 milliseconds long. To prevent aliasing effects

at the segment boundaries the frames usually overlap (e.g. the following frame starts in the middle of the current frame) and some window function is applied (e.g. the Hamming window). This section will give a brief overview of the most common methods of feature extraction for speech recognition, a more detailed description can be found e.g. in [Hun99].

Today's standard is to use features based on frequency distribution in the speech frame. For example a bank of band pass filters (originally analog devices but now computed by the means of discrete Fourier transform) can output a feature vector of short time energy values for each specified band. Based on the observation that human ear is more frequency-sensitive at lower frequencies the most successful filterbank analysis for speech recognition uses bands that are equally spaced up to 1 KHz and logarithmically spaced bands beyond this frequency (this is referred to as *technical mel-scale*, see [Hun99]).

Another approach, the *linear predictive coding* (LPC), models the vocal tract as an all-pole filter with its parameters computed by a fast *autocorrelation method*. It can be viewed as a smooth parametric fit to short time spectrum of the speech signal. Although its computation is faster than one of a filterbank analysis the recognizers based on this method have recognition accuracy inferior to recognizers with filterbank computed features.

Unfortunately the coefficients obtained by both mel-scale filterbank analysis and LPC are highly correlated. The use of Gaussian mixture acoustic model with diagonal covariance matrix requires the features to be uncorrelated and it has been observed (see e.g. [Teb95]) that the use of uncorrelated features as the input of a neural network acoustic model increases its performance. Fortunately it turns out that the application of inverse cosine transform to the logarithms of the outputs of a filterbank yields to coefficients that are close to be uncorrelated. Its result is often called a *cepstrum* leading to a standard for features in speech recognition called *mel-scale filterbank cepstral coefficients* (MFCC).

Perceptual Linear Prediction was introduced by Hynek Hermansky [Her90] as a novel approach to feature extraction based on human hearing properties. It makes an all-pole fit (as in LPC) to the outputs of a mel-scaled filterbank and also applies a loudness sensitivity curve to the spectrum respecting the reduced sensitivity of human ear at the low and high ends of the spectrum. It is often combined with so called RASTA filter (see [HM91]) which tries to compensate for distorting effects of the communication channel.

4.3 Phonetic Units

In continuous speech recognition it would be unfeasible to have models of whole words or even sentences. Instead the words are usually composed of some kind of sub-word units. These correspond to classes of which probabilities are being estimated during the acoustic modeling phase and will be further referred to as *phonetic units*.

The simplest kind of phonetic units are called *monophones* and roughly correspond to phonemes[2] of the given language. Their advantage is their relatively small number (for example 36 monophone units were used in [Pav03] for the recognition of Czech sentences) and thus lead to low computational costs and require less training data.

In speech the acoustic realization of a phoneme is much dependent on its *context*, i.e. the surrounding phonemes. It has been observed by many (see [Ode95] and [Ser97] for further references) that explicit modeling of the context dependency can lead to superior performance of the recognition system. Hence today's state-of-the-art recognition systems operate with *context dependent* phonetic units (as opposed to context independent monophones). Among context dependent units belong:

- **Biphones**. Can be either left, depending on the identity of the preceding phoneme or right, depending on the identity of the following one. An interesting observation by [Ser97] is that when trained and tested on the same data the recognition performance of right biphones is different than one of left biphones.

- **Triphones**. The most popular context dependent units where each unit is modelled as being dependent on both the preceding and the following unit.

- **Syllables**. Used in so called segment-based systems where the segmentation is non uniform as opposed to frame based systems and segment boundaries must be found prior to acoustic modeling. Syllable boundaries are easier to find than boundaries of phonemes. See [Hu96] for details.

- **Word-dependent units**. The context of a word dependent unit is the word in which it occurs. Rather than have a whole word-dependent recognition system (which would lead to extremely large number of

[2]The most common definition of a phoneme is that it is the smallest unit of speech that distinguishes one word from another.

phonetic units) the word-dependent models may be used only in a limited number of "problematic" words while the rest of the words are composed of some standard type of context dependent units. An example (see [Ser97] for details and further references) may be the function words in English such as *of*, *the* or *with*.

- **Senones**. In HMM systems both monophones and context dependent units are often composed of more (usually three) HMM states. The speech recognition system Sphinx [Hua92] uses the individual states of the HMM as phonetic units called *senones* leading to recognition performance superior to biphones and triphones.

- **Polygraphs**. These phonetic units are based on the orthographic rather than phonetic representation of a word. The main idea is that the phonetic realization of a grapheme (i.e. a letter) can be determined given a sufficiently large context of surrounding graphemes. It has been shown in [Sch93] that polygraphs can lead to a performance comparable to triphones without the need for having phonetic transcriptions of words in dictionary.

Depending on the availability of training data and computational resources some compromise must be made between granularity (the higher the number of phonetic units the more trainable parameters and training data is required) and context sensitivity (higher number of phonetic units means more precise modeling). Different specimens of context dependent units have different relative occurrences in speech data and for some there will be insufficient training data. This is referred to as sparse data problem.

One strategy that deals with this problem is *parameter tying*. If two context dependent units are acoustically sufficiently close, their parameters can be tied, i.e. they will share the same acoustic model (e.g. Gaussian mixture or output neuron). The decision on which phonetic units to tie can be based on expert knowledge or derived from the training data. A method for automatic parameter tying using decision trees is described in detail in [Ode95].

Another strategy called *backing-off* (details may be found in [Ser97]) uses several levels of context dependency. The levels consist of monophones, biphones and triphones trained on the same data. If, during recognition, an ill trained triphone (due to lack of data) is to be used, the system may *back-off* to its biphone or monophone counterpart.

Context dependent units are today's standard in HMM based systems. Neural network based ones benefit from *context sensitivity* of their monophone models which will be described in next chapter.

4.4 Neural Network Acoustic Model

Ever since the interest in neural networks has been renewed after the discovery of the backpropagation algorithm there have been various attempts at their use for the task of speech recognition. In [HL87] is described an experiment, where a multi-layer perceptron is trained to form decision regions in two dimensional space formed by the first two formants of English vowels. Multi-layer perceptron has been applied to recognition of isolated digits with its performance exceeding one of an HMM system [PM88].

A whole neural network approach to continuous speech recognition is a difficult problem since the number of inputs and outputs varies with every recognized utterance. Although there have been successful attempts (see multi-state time delay neural network in [Teb95]) they lack the flexibility offered by hidden Markov models. For example if a new dictionary is to be used, the whole network needs to be retrained.

Today's most successful solution is referred to as a *hybrid approach* (described by [BM97], [Coh92], [Sut98], [Rob94] and many others) where the neural network estimates the emission probabilities of a hidden Markov model.

It has been shown in section 3.1.2 that a multi-layer perceptron is capable of learning any arbitrary function as well as output class posterior probabilities. If each output of the network is associated with a HMM state S_j then the activations of output layer neurons can be interpreted as probabilities $P(S_j|o)$ which can be by the application of Bayes' rule changed to state emission probabilities:

$$P(o|S_j) = \frac{P(S_j|o) \cdot P(o)}{P(S_j)}. \tag{4.1}$$

The term $P(o)$ remains constant during the whole recognition process and hence can be ignored so the emission probabilities can be acquired by dividing the network outputs by the class priors. According to the proponents of hybrid approach (see e.g. [BM97], [Teb95])The usage of a neural network has many potential advantages over the standard Gaussian mixtures:

- **Model accuracy**. The use of Gaussian mixtures requires some assumptions on the form of the probability distribution being modeled such as that the features are statistically independent. On the other hand a multi-layer perceptron does not place any such assumptions on the data and can approximate any function. This even allows to join different feature types on the input of the classifier, e.g. binary and real numbers.

- **Context sensitivity**. If several adjacent feature vectors are appended to form the input of the neural network $X_{t-c}^{t+d} = \{x_{t-c}, \ldots, x_t, \ldots, x_{t+d}\}$ or if recurrent neural networks are used then time correlation of the feature vectors can be taken into account when estimating the probability distribution. Gaussian mixture based systems also try to incorporate contextual information by adding first and second order derivatives to the feature vector (see e.g. [You02]) but the neural approach is obviously more general.

- **Economy**. Gaussian mixtures use their parameters to model the surface of the density function in acoustic space, in terms of likelihoods $P(input|class)$, while neural networks use their parameters to model class boundaries, in terms of posteriors $P(class|input)$. Boundaries require less parameters and thus can make a better use of limited training data. Empirically ANN/HMM hybrids require less trainable parameters to obtain the same accuracy as conventional HMM systems. It has also been observed by [RH95] that the availability of posterior probabilities allows for a more efficient pruning during the decoding phase.

- **Discrimination**. As has been said in section 2.4 the standard training criterion of HMMs does not guarantee discrimination between models. Neural networks, if trained with SSE or similar error criterion, easily provide discrimination although on local (frame) level only.

4.5 Training

An important issue for training of a speech recognition system is the availability of phonetic labels for the training data. Section 2.3.2 describes how parameters of an HMM can be estimated with only phonetic transcription of the training utterances. Prior to embedded training some initial estimates of the HMM parameters must be chosen.

A successful strategy referred to as *flat start* first creates one prototype of a HMM representing a phonetic unit with (in the case of Gaussian mixture acoustic model) global means and variances computed on all the training data. All the models of phonetic units to be trained are then assigned the parameters of the prototype. The flat start procedure expects that each training utterance is uniformly segmented. The hope is that enough of the models will align with their actual realizations so that in the following iterations of embedded training the models will align as intended [You02].

For the training of a neural network each frame in the training set must carry information (label) about which phonetic unit does it represent. If

an already trained speech recognizer is available (regardless whether HMM based or ANN/HMM hybrid) a technique called *forced Viterbi alignment* may be used to automatically generate the labels. Similarly to embedded training a composite HMM is constructed based on phonetic transcription of the processed utterance. If the Viterbi algorithm (see sec. 2.2.2) is run with this composite HMM on the utterance, the result is the single most likely state sequence carrying information about which state generated which speech frame (observation vector). If the states correspond with the output neurons of the network the labels of each frame are known.

If a network is trained by automatically generated labels it can be used as an acoustic model in a hybrid ANN/HMM system and forced Viterbi alignment applied to provide yet another generation of labels. It has been observed (e.g. by [Teb95]) that repeating this process several times can further improve the recognition accuracy. This technique is referred to as *recursive labelling*.

The most common weight update strategy used when training multi-layer perceptrons is the incremental version of the backprop algorithm. The desired network output (see equations 3.6 and 3.17) is constructed by setting to one the desired value of the neuron corresponding to the phonetic unit in the label, all the other desired neuron values are set to zero. An alternative to this approach can be setting the desired neuron values to their corresponding state probabilities $\gamma_t(i)$ computed from the forward and backward variables (eq. 2.34) instead of the Viterbi algorithm. This is a more soft decision about which phonetic unit does a particular frame belong to carrying additional information about the certainty of the categorization of the frame.

4.6 Decoding

The retrieval of most likely word sequence $W^* = w_1, w_2, \ldots, w_n$ from a sequence of observation (feature) vectors $O = o_1, o_2, \ldots, o_T$, sometimes called *the fundamental problem of speech recognition* can be in the most general form expressed by the equation

$$W^* = \underset{\forall W}{\operatorname{argmax}} P(W|O) = \underset{\forall W}{\operatorname{argmax}} \frac{P(O|W)P(W)}{P(O)}. \tag{4.2}$$

In this equation the term $P(O|W)$ is the acoustic model likelihood of the word sequence and the term $P(W)$ is a prior probability of the sequence computed by so called *language model*. The term $P(O)$ can be excluded from the computation as it is independent of the word sequence.

Obviously the evaluation of the probability $P(W|O)$ for each possible word sequence is not computationally feasible except for the case of isolated word recognition, where it is possible to evaluate the probability of each word in dictionary and choose the one with the highest. In continuous word recognition the task of the decoder is to search for the most likely word sequence in an efficient manner.

4.6.1 Language Model

Before going into details of the decoding algorithm let's first discuss the issues of language modeling. The language model is responsible for the computation of the prior probability of the word sequence $P(W)$. Generally there are two kinds of language models: *deterministic* and *stochastic*.

In deterministic language models the probability of the word sequence W may have only two values, one or zero, i.e. the sequence is either possible or not possible. A simple example of a deterministic language model is the *word pair grammar* where each word in the dictionary has associated a list of words which can follow it.

A more powerful deterministic language model can be a finite state automaton with words associated with the state transitions. If a word sequence is accepted by this automaton its probability is set to one, otherwise it is set to zero. The advantage of this approach is that the automaton can be derived from a hand written non-recursive context free grammar (such as the Extended Backus Naur Form (EBNF), see [You02]).

Stochastic language models try to actually estimate the probability of the word sequence and usually have trainable parameters. The most commonly used are *N-gram language models* where the probability of the word sequence is expressed as

$$P(w_1, w_2, \ldots, w_N) = \prod_{i=1}^{N} P(w_i|w_1, w_2, \ldots, w_{i-1}). \tag{4.3}$$

The simplest N-gram model for $N = 1$ is referred to as *unigram language model* (for $N = 2$ it is bigram, for $N = 3$ trigram and so on...). The higher order language models suffer from the sparse data problem (the higher the N the more text data is needed to train the model) and thus strategies to compensate for ill-trained N-grams must exist. See [JM00] for more information on this matter.

4.6.2 Time Synchronous Decoder

The Viterbi algorithm described in section 2.2.2 is an example of an efficient decoding algorithm. In order to apply it to the task of continuous speech recognition a composite HMM representing all possible utterances to be recognized by the system. To do that lexical and language model must exist. The lexical model is a pronunciation dictionary expressing which HMMs representing phonetic units does a given word consist of. The language model defines allowed transitions among those words, optionally, if it is a stochastic language model, the probabilities of those transitions.

Besides storing the previous best state backpointer $\psi_j(t)$ (equation 2.25) it is useful to also store a backpointer to previous best word together with the frame that word ends in. This simplifies the retrieval of the most likely word sequence at the end of the search.

Note that the Viterbi algorithm is suboptimal in a sense that it does not maximize the probability of the word sequence as expressed in equation 4.2. Instead it maximizes the probability of a single most likely path along the HMM states as described in section 2.2.2. This has been shown to work well in practice and unlike the forward procedure (section 2.2.1) it allows for the retrieval of the word sequence.

The Viterbi algorithm is only suitable for simple language models, such as bigrams or word pair grammars since the probability of the next word is only dependent on the current word regardless of longer history and thus satisfies the first order Markov assumption. It is also possible to use finite state automaton as a language model simply by replacing its states by composite word HMMs. If long-span language models such as trigrams were to be used in the Viterbi search multiple copies of word models would be required to account for all possible contexts. That would be computationally unfeasible even for small vocabularies.

4.6.3 Pruning

Although the exhaustive search by Viterbi algorithm guarantees to find the single most likely path it spends a significant portion of the computation time evaluating unlikely paths. Pruning strategies try to omit computations for paths with low probability. The most common technique is called *beam search*.

During the processing of a frame the state with highest score is found. The idea behind beam search is to expand (i.e. take into account during computations for the next frame) only those states that have score higher than a chosen proportion of the highest score. The beam width, i.e. the

constant expressing this proportion, needs to be determined empirically. If pruning is done the search no longer guarantees to find the most likely word sequence so a compromise must be made between the speed and the accuracy of the search.

4.6.4 Best First Decoder

The decoding algorithm discussed in the previous section is an example of a breadth-first search where all the search variables for the current frame are computed before moving to the next frame, hence the name time synchronous search. In an asynchronous search the best scoring hypothesis is expanded first regardless of time. This is the alternative to the Viterbi algorithm in many large vocabulary recognition systems (see e.g. [Ode95], [RH97]).

The best-first search is based on the heuristic A* search used in artificial intelligence. The search uses a composite function to evaluate the score $f_h(t)$ of each hypothesis h at time t:

$$f_h(t) = a_h(t) + g_h^*(t) \tag{4.4}$$

where the term $a_h(t)$ is the score of the partial hypothesis based on the information collected to time t and the term $g_h^*(t)$ is a heuristic expectation of the reminder of the score up to the end of the utterance. If the reminder of the score $g_h^*(t)$ is an upper bound on the actual score of the hypothesis, the search is admissible, i.e. guarantees to find the same best path as an exhaustive search.

Time asynchronous search has many potential advantages over the synchronous Viterbi algorithm, namely:

- The most likely word sequence can be found before all possible hypotheses are evaluated saving computation time.
- It is possible to use higher order language models during the search.

The main disadvantage of this technique is its sensitivity to the chosen heuristic $g_h^*(t)$ (see [RH97] for more information on heuristic functions).

Chapter 5

Experimental Setup

5.1 Performance Measures

5.1.1 Word Level Accuracy

To analyze the word-level performance of the tested recognition system the recognized sentence is matched against the referential (i.e. correct) sentence by dynamic programing. The method is similar to computing the Levenshtein [Lev66] metric for strings. The output of the algorithm is the following set of numbers: N - the total number of words in the referential sentence, H - the number of "hits" (correct words) in the recognized sentence, S - the number of word substitutions, I - the number of word insertions into the referential sentence, and D the number of word deletions from the referential sentence needed for the two sentences to match.

In order to synchronize our scoring results with the HTK scoring package the following weights were used: 7 for insertions and deletions and 10 for substitutions. It is also possible to define equivalence classes for words. Those words that belong to one class are considered equal during the match, i.e. are counted as hits.

Having the numbers from the match these two measures are computed:

$$\%Corr = \frac{N - S - D}{N} = \frac{H}{N} \tag{5.1}$$

$$Acc = \frac{N - S - D - I}{N} = \frac{H - I}{N} \tag{5.2}$$

The first one (5.1) is simply the percentage of correctly recognized words. It can happen (and is often the case) that the recognized sentence is significantly longer than the referential one. Even though such sentence might have a high $\%Corr$ score it also contains many extra words which can be considered to be

an inferior performance factor. The second measure (5.2) tries to compensate for this by subtracting the number of such words (which are counted as insertions by the matching algorithm).

5.1.2 Neural Network Training

Mean Square Error (MSE) The error function being minimized during training is the summed square error E^p (see eq. 3.6). The overall network error for the whole training pattern is computed as

$$MSE = \frac{1}{T}\sum_{p=1}^{T} E^p, \tag{5.3}$$

where T is the number of training patterns (the number of speech frames).

Frame Error Rate (FER) When a speech frame is processed through the network and the output unit with the highest activation does not correspond with the correct phonetic unit, a *frame error* occurred. The frame error rate is

$$FER = \frac{\text{no. of frame errors}}{\text{no. of frames}} \tag{5.4}$$

computed for the whole tested pattern.

5.1.3 Time

The speed of speech recognition applications is usually measured in real-time percentages:

$$\%RT = \frac{\text{time to run}}{\text{total time of testing data}} \tag{5.5}$$

The referential machine for time measures was a Pentium4 3.2 GHz.

5.1.4 Average Number of Active States

An active state is a HMM state in a given speech frame that has not been pruned during the decoding search (see sections 8.3.1 and 8.4.1). An important measure which can provide information about the computational expense of the search is the average number of active states per speech frame:

$$\text{avg. no. of active states} = \frac{\text{total no. of active states}}{\text{total no. of speech frames}} \tag{5.6}$$

5.1.5 Confidence Interval

It must be taken into consideration that the testing sample is always finite and we must accommodate for random errors. To do this we introduce a confidence interval that will be displayed together with the test measures. For data that express proportions the *binomial proportion confidence interval* can be used with the most common (see e.g. [Nis07]) approximation

$$p \in (\hat{p} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}) \tag{5.7}$$

where p is the correct proportion (e.g. $\%Corr$), $\hat{p}$ is the measured proportion, n is the sample size (i.e. the total number of words) and z is a percentile of a standard normal distribution corresponding to the desired percentage confidence.

The problem with binomial distribution is that it assumes statistical independence of individual errors of which the proportion (i.e. error rate or accuracy) is computed. This is clearly not true in speech recognition where each error can have influence on subsequent errors. We can, however, assume independence of errors in different sentences since all search variables are discarded after a sentence is recognized and there is no way a recognition of one sentence can affect recognition of the next one.

In [Vav08] we have investigated the statistical properties of a sample of independent observations $(r_1, s_1), (r_2, s_2), \ldots, (r_n, s_n)$ where (in the case of speech recognition) r_i is the number of correctly recognized words in sentence i and s_i is the number of incorrectly recognized words. We have derived the distribution function of the variable

$$\xi_n = \frac{\sum_{i=1}^{n} r_i}{\sum_{i=1}^{n} r_i + \sum_{i=1}^{n} s_i} \tag{5.8}$$

which can be used to find the boundaries of a confidence interval. The details of the boundary computation can be found in appendix B. See [Vav08] for a more comprehensive discussion.

Note that the confidence interval computation can be done for both $\%Corr$ and Acc. If we label N_i, H_i, S_i, I_i and D_i the results of a dynamic programing match of a single sentence i then for the computation of $\%Corr$

$$r_i = H_i, \tag{5.9}$$

$$s_i = N_i - H_i. \tag{5.10}$$

For computation of the measure *Acc* we can set

$$r_i = H_i - I_i, \tag{5.11}$$

$$s_i = N_i - H_i + I_i. \tag{5.12}$$

5.2 Training Corpora

All the available speech data is in Czech language, recorded in quiet environment at 16 KHz sampling rate and 16 bits per sample. The corpora are divided into sentences; each sentence is stored in a separate file. The training set consists of three parts:

- **Train Schedule Queries**. This corpus consists of questions about train schedules and related information. An example of such question would be "When does the train for Plzeň leave" or "At what time does the fast train leave from Písek to Prague". The corpus originally contained 100 speakers (50 male, 50 female) but some of the files were broken and could not be used.

- **LAC-HP Chess**. Stands for LASER Audiocorpus High Precision. The corpus was recorded in an audio studio, all the recordings have been verified by members of the LICS team. This set consists of voice commands for a chess game. The commands could be either chess moves (e.g. "Move the king to b5") or miscellaneous commands like "I want to start a new game".

- **LAC-HP Phonetic**. This is a set of nonsense sentences with words containing infrequent phonetic units. Among the least frequent phonetic units are for example o:, u:, z' or t'.

Corpus	**Speakers**	**No. of sentences**	**Vocabulary size**	**Total Length**
Train Schedules	81 (48M, 33F)	11270	1490	11:28:06
LAC-HP Chess	81 (31M, 50F)	2050	96	1:51:50
LAC-HP Phonetic	81 (31M, 50F)	1200	115	1:33:02

Table 5.1: Training corpora

5.3 Testing Corpora

Three sets were chosen for the purpose of testing. The first two are from the same domain as the training data, the last one (numbers) is used to test the performance on a domain which was not present in training data[1].

- **Train Schedules.** This is just a subset which was taken out of the trains corpus. These sentences were not present in the training data.
- **Chess Moves.** It was discovered during the early stages of testing that commands unrelated to chess moves are much easier to recognize than the move commands (mainly because of the column names). The recognition accuracy of commands such as "start a new game" was very close to 100%. Those commands were removed from the test corpus to make the results more easily comparable.
- **Numbers.** This corpus consists of spoken numbers between one and one million. The numbers were first chosen by a random number generator and then converted to text form and recorded. No part of the numbers corpus was included in the training set.

One problematic part in both moves and numbers corpora were the numbers seven and eight. These can be pronounced a either "sedm" (seven) and "osm" (eight) or as "sedum" and "osum". The first pronunciation is standard Czech, the second colloquial Czech. During the speakers were not properly instructed which pronunciation to use which makes the recognition rather problematic (and it has a significant effect on recognition accuracy). For this reason these pairs were placed in equivalence classes (see section 5.1.1) during performance scoring.

Corpus	**Speakers**	**No. of sentences**	**Vocabulary size**	**Total Length**
Train schedules	4 (2M, 2F)	400	1490	0:31:34
Chess moves	20 (10M, 10F)	2000	96	1:18:28
Numbers	28 (28F)	1399	40	1:29:08

Table 5.2: Testing corpora

[1]The testing set also contains different speakers than the training set.

5.4 Feature Extraction

While the performance of a speech recognition system can be greatly affected by the choice of feature extraction methods we have decided that proper testing of at least the most common possibilities would be beyond the scope of this work. Besides the standard MFCC some experiments were carried out with RASTA parametrization. Unfortunately the results were much worse than those of the MFCC. RASTA parametrization has several parameters which can be tuned and we suspect that the performance that we have achieved might have been poor due to a suboptimal setting of these parameters.

We have decided to use only the Mel-Frequency Cepstral Coefficients (MFCC) during all our subsequent tests. Below is a list of the parametrization settings. Some of those were experimentally tested, some were just taken from the litereature or samples from other ASR software (such as HTK [You02]).

- Features for one speech frame were computed from 32 milliseconds of speech (512 samples at 16KHz). The frames overlap by 50%, i.e. 16 milliseconds. This has proven to work well with both trains and chess corpora. The other setup which was tested, 16 ms window and 8 ms overlap, did not lead to performance increase on these corpora (but increased computational costs). The only shortfall of this setting has been the Numbers corpus: The three state phonetic unit models used in most of our tests meant that minimal phoneme duration constraints were too high for this corpus (the numbers were spoken too fast) which led to performance degradation. See section 8.2.3 for discussion and possible workarounds.

- The total number of computed MFCC coefficients was 24 but only the first 12 are used, the rest is discarded. Tests were performed with all 24 coefficients, but this did not lead to any performance increase.

- The preemphasis coefficient is 0.97, Hamming window is applied before the FFT transform of the speech signal.

- Cepstral liftering (in a way similar to the one described in [You02]) is applied.

- Short time energy is usually used instead of the 0th cepstral coefficient. Our system does not compute short term energy, the 0th cepstral coefficient is kept.

5.4.1 Postprocessing

The segment of speech signal (frame) from which features are computed is very small and it has been experimentally proven that the augmentation of these features by some information from the neighboring frames can significantly improve the recognition accuracy (see section 7.1.1 for some of those experiments).

In the case of GMM based acoustic models the problem is (if only diagonal covariance matrices are used) that all the components of the feature vector must be uncorrelated. This can be solved by adding so called *delta* and *acceleration* coefficients which are computed in the following way (taken from [You02]):

$$d_t^i = \frac{\Sigma_{\theta=1}^{\Theta}\theta(c_{t+\theta}^i - c_{t-\theta}^i)}{2\Sigma_{\theta=1}^{\Theta}\theta^2} \tag{5.13}$$

where $\Sigma_{\theta=1}^{\Theta}$ is the ith component of the delta coefficient vector at time t, c_t^i is the ith component of the computed feature vector in time t. The parameter Θ controls the size of the context window, in all our experiments we have set the value of Θ to 2. Once the delta coefficients d_t are computed the same formula is applied to them to compute the acceleration coefficients. This means (for $\Theta = 2$) that information gathered from 9 subsequent frames is used as the input for the GMM acoustic model.

In the case of a neural network acoustic model there is no need to insure that the components of the input vector are uncorrelated and subsequent feature vectors can be simply combined into one:

$$C_{t-\Theta}^{t+\Theta} = \{c_{t-\Theta}, \ldots, c_t, \ldots, c_{t+\Theta}\} \tag{5.14}$$

The parameter Θ controls the number of frames to be added to each side of the current frame.

The features added by postprocessing can cause problems during live recognition because frames from the future and from the past of the current frame are needed for the computation. This needs to be addressed at the beginning and at the end of the recording, because such frames are obviously not available. There are several possible solutions, for example the frames which do not have the sufficient number of predecessors (resp. successors) can be discarded. Other solution is to copy the nearest available vectors sufficient times as to provide enough input for the postprocessing. Our solution is to accumulate the feature vectors until sufficient number of neighboring frames is available and then proceed with the computation. The entire output (e.g. the frame + delta + acceleration) is copied from the nearest available frame for those frames that do not have the sufficient number of neighbors.

5.5 Alphabet

As described in section 4.3 the word HMM models in an automatic speech recognizer consists of concatenated phonetic unit models. Regardless of the type of these units (e.g. monophones, triphones, etc.) a set of symbols for phonetic transcription of words must be defined. The most universal way of phonetic transcription is provided by International Phonetic Alpahbet (IPA, [IPA99]) which includes all the phonemes used in the Czech language. It is rather impractical for computer processing since it requires a special font just to be displayed. For this reason the Speech Assessment Methods Phonetic Alphabet (SAMPA) has been designed in which all the phonemes can be written as sequences of ASCII characters. Our alphabet is based on the Czech SAMPA [Bat03] with the following exceptions:

- The symbol for long vowels (:) has been changed because at one point of system design the phoneme models had to be stored in files having the same name as the phonemes.
- When phonetic transcriptions were done by inexperienced users there were many errors in phonemes with less obvious transcriptions. For this reason some transcriptions were changed to more intuitive (for Czech speakers) forms. For example `J\` was changed to `d'`.
- Due to small size of our speech corpora some of the less frequent phonemes were left out.

The complete alphabet can be found in appendix A.

5.6 LASER Recognizer

The LASER (LICS Automatic Speech Extraction/Recognition) recognizer has been under development by Laboratory of Intelligent Information Systems (LICS), University of West Bohemia since 2001 (see [Pav03], [EP04a], [Pav06]). Originally each of the modules was a command line program written in C or Pascal. Both inputs and outputs of each module were stored in files.

The core modules are: **LRec**: The recorder, **LPar** – The feature extraction module, **LNNM** – The neural network module, **LGMM** – The Gaussian mixture module, and **LDec** – The Decoder (see section 4.1 for a list of typical ASR components).

Having the modules as separate command line tools had several problems:

- During live recognition the starting and stopping of the recording had to be implemented in the recording module, information from other modules could not be used.
- The whole utterance must have been recorded and saved before the other modules could start to run.
- The loading and saving of input and output files for each of the modules could slow down the recognition.
- The files had to contain all the information necessary for the next module. For example the acoustic models (LNNM or LGMM) had to compute probabilities for all the phonetic units for the frame being processed. In the case of context dependent phonetic units this means computing and saving large arrays of probabilities which may not all be needed during the decoding phase, since the decoding is pruned.

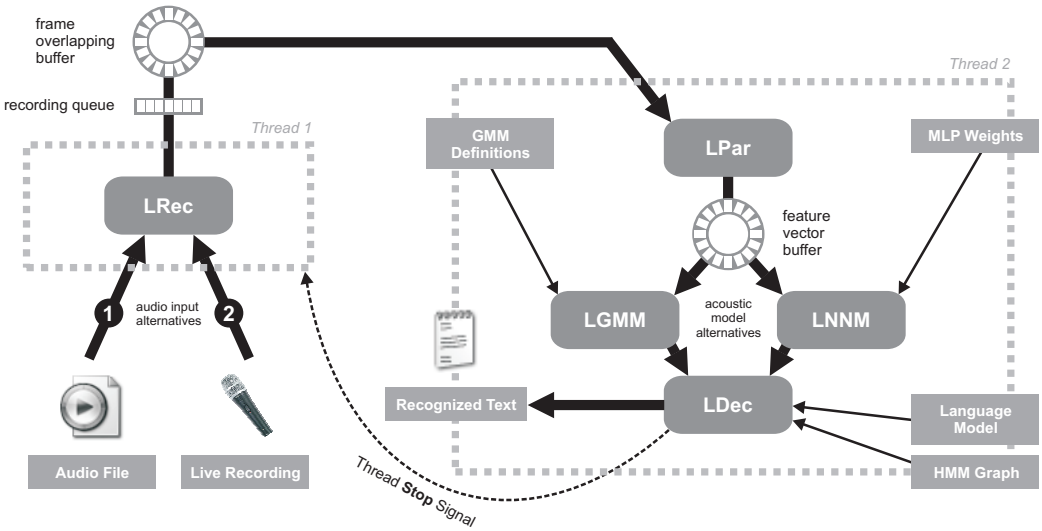

Figure 5.1: Architecture of the JLASER speech recognition system.

At the beginning of 2007 we have experimented with Java and implemented the neural network acoustic model and the decoder. The initial tests have shown that the drop in recognition speed was lower than expected and soon the other modules were implemented as well. Due to changes in design the new recognition system named JLASER ran faster than the original recognizer. This is similar to what was observed during the development of the Sphinx 4 [Wal04] recognition system where the implementation in the Java

language boosted the development of new algorithms and the new system attained higher speed than the previous version written in C.

In comparison with the earlier implementation the JLASER system benefits from several features of the Java programming language, namely:

- The language is platform independent. The software has been tested on Windows and Linux without the need to change any part of the code.
- The platform APIs deliver a number of useful tools such as hash maps, XML parsing, etc. which reduces the need for additional coding.
- Object oriented programming provides an effective way for modular design. Each of the main ASR components can have several implementations which can be transparently accessed through polymorphism.
- The garbage collector eases the programming task because the developer does not have to worry about memory leaks and can concentrate on the design of algorithms. Not only that, garbage collection also allows a more effective management during decoding: Only the most recent active list (see section 8.3.1) needs to be stored, all the paths containing the states from the previous frames can be accessed via the backpointers. If a state is pruned from the active list, the entire path can be discarded from the memory. Garbage collection allows doing this at times when memory is full.
- There are other projects by the LICS group (see e.g. [Hab08]) which are implemented in Java so a common programming language enables an easy integration.

The main trade off for the advantages of a high level language such as Java is usually speed. Besides comparing our old implementation of the recognizer with the JLASER we have also carried out tests with HTK recognizer. The results were different for each testing corpus, but the worst case was a 93% relative increase in recognition time for the train schedule corpus. It should be noted that the experiment was done with a linear lexicon since HTK does not support tree organized lexicons. When a tree lexicon was used JLASER outperformed HTK by 46% relative. The details of these experiments can be found in [Pav07].

Chapter 6

Gaussian Mixture Acoustic Models

A Gaussian mixture model (GMM) estimates the emission probability $P(o|S_j)$ of an observation o (a feature vector computed from one speech frame) for a given state S_j by computing the equation 2.8. Even though the official name of the model is *mixture of Gaussian functions* where the mixture term refers to equation 2.8, the individual components (computed by equation 2.9) are often called "mixtures". To avoid confusion it should be stated that in this work a mixture refers to a single multivariate Gaussian function computed by eq. 2.9 and we can for example talk about "adding mixtures" during the training of the models.

The following information relates to all the models discussed in this chapter:

- Each phonetic unit model (whether it is monophone or triphone) consists of three states. Each of these states has its own GMM acoustic model. In the case of triphone models some states may share the same acoustic model due to parameter tying.
- Only models with diagonal covariance matrices were tested. This places a constraint on the form of observation vectors requiring the individual components of the vector to be statistically independent. This is assured by the nature of MFCC coefficients and by adding delta and acceleration coefficients, see sections 4.2, 5.4 and 5.4.1.
- All the GMM models were trained by the Hidden Markov Toolkit (HTK, see [You02]) but the testing was done with the JLASER (see section 5.6) recognizer.

6.1 Monophone Training

The GMM models for all the phonetic units are initialized by a so called flat start strategy: The global mean and variance vectors are computed from the training data, and these are used as parameters for a single Gaussian mixture. This mixture is then cloned to construct acoustic models for all phonetic units.

Afterwards the parameters of the models are estimated by embedded training (see section 2.3.2) with the Baum-Welch algorithm. Several iterations of the re-estimation procedure are performed.

The number of trainable parameters (and hence the expressive power) of the model can be increased by adding mixtures. We use the following scheme for training and adding mixtures:

1. The process starts with trained single mixture models.
2. The number of mixtures in each model is doubled.
3. The new models are trained by several iterations of the Baum-Welch algorithm.
4. Steps 2 and 3 are repeated until either the increase in the number of mixtures does not lead to a significant increase in accuracy measured on the testing data, or the resulting model is too slow due to a large number of mixtures.

The number of mixtures is increased by *mixture splitting*: the mixture with the highest weight component (c_{jm} from eq. 2.8) has the weight decreased by half, is cloned and the means of the newly created mixtures are shifted (one by subtraction, one by addition) by 0.2 standard deviations. This is repeated until the desired number of mixtures is reached. The splitting algorithm also deals with so called defunct mixtures, i.e. those with very low weights, the details can be found in [You02].

6.2 Decision Tree Clustered Triphones

The number of monophones is relatively low (36 in our case, see section 5.5) and sufficient amount of training data is available to properly estimate the parameters of each of the GMM models. Unfortunately this is not the case when triphones are introduced and the need to compensate for sparse data arises. The problem is solved by parameter tying, i.e. by sharing the same Gaussian mixtures among several states.

There are various ways of how to automatically tie the state models based on the training data. Only a brief overview will be given here; our experiments with triphone models are described in detail in [Hej07]. Two approaches to automatic state clustering were tested:

- **Data driven clustering.** A metric that describes the distance between two states is introduced (see [You02] for details). The states are initially placed each in its own cluster. Afterwards the two states that form the smallest cluster are put into one cluster. The size of a cluster is defined as the greatest distance found between two states in the given cluster. This is repeated until a stopping condition is reached which can be either that the total number of clusters reached a threshold or the size of the largest cluster exceeds a threshold.

- **Decision tree clustering.** A collection of phonetic questions with a yes/no answer is created (e.g. "Is the left context of the triphone a vowel?", see [Hej07] for a complete list of questions in our experiments). Each question splits the set of states into two parts. A metric exists that estimates how "good" this split was (see [Ode95],[You02]). The decision tree is constructed in the following way: The question with the best score is placed in the root node. After that the same is done recursively for the child nodes until a stopping condition is reached. Each leaf node then holds a cluster of states that share the same GMM model. This way a GMM model can be found for any triphone, even for those that were not seen in training data.

Besides the fact that the decision tree clustered triphones are more universal (models for triphones not seen in the data can be easily constructed) the experiments described in [Hej07] have shown that decision tree clustering leads to higher recognition accuracy than data driven clustering. Table 6.1 shows the achieved results together with the numbers of physical state models.

Threshold	Number of state models	Best results	
		Train Schedules	Chess Moves
1000	1733	84.46 @ 16mix	97.55 @ 16mix
2000	1070	83.93 @ 32mix	97.51 @ 32mix
5000	517	82.46 @ 32mix	97.52 @ 32mix

Table 6.1: Achieved $\%Corr$ for different values of decision tree clustering threshold.

Here is the whole training process in detail:

1. We start with trained single mixture monophone models. These are cloned to create triphone models for all the triphones seen in the training data. No cross-word triphones are created, the ones lacking the appropriate context are replaced by diphones (beginnings and ends of words) or monophones (single phoneme words).

2. The triphones are trained by several iterations of the Baum-Welch algorithm.

3. The trained triphones are clustered and the newly created tied-state models are further trained.

4. The number of mixtures for each state is increased in the same way as in the case of monophones.

The results for both monophone and triphone models can be seen in Figure 6.1 which depicts a comparison of the different models based on performance expressed in terms of recognition accuracy ($\%Corr$) and recognition speed ($\%RT$).

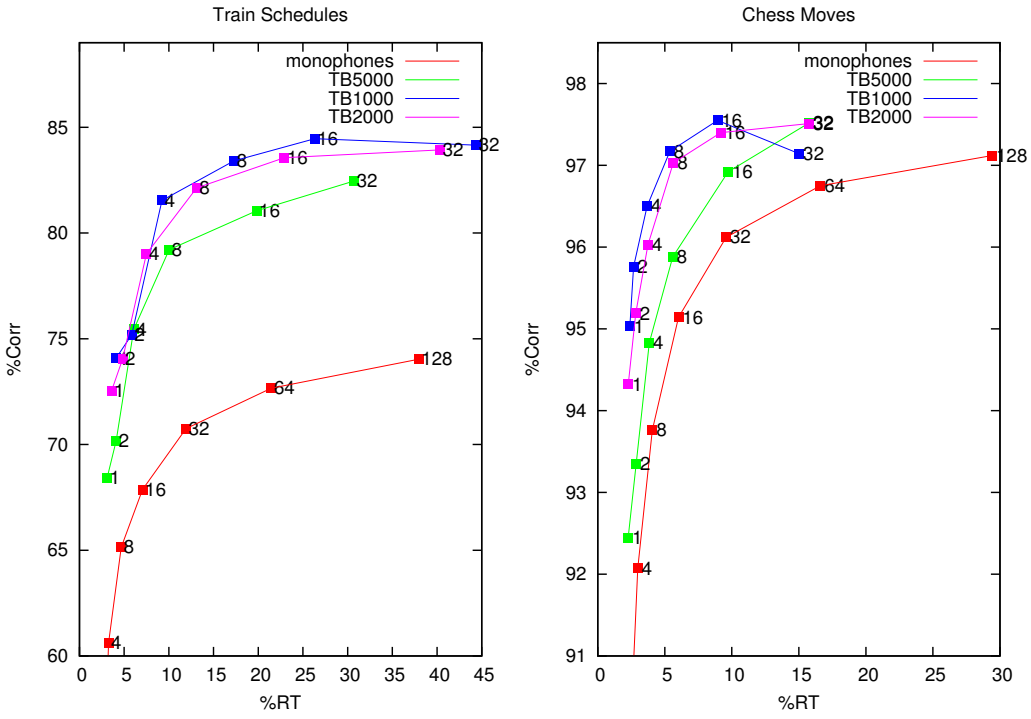

Figure 6.1: Results for different numbers of mixtures per state.

Chapter 7

Neural Network Acoustic Models

There are many neural network types that have been tried for automatic speech recognition such as time-delay neural networks (see e.g. [Teb95]) or recurrent neural networks (e.g. [Rob89]) but the most widely and successfully applied architecture is the multi layer perceptron. For this reason we have decided to start our experiments with this architecture. As this chapter will show the multi layer perceptron as an acoustic model comes with a rich variety of design possibilities and we have chosen to explore these thoroughly rather than try different types of neural networks.

Before getting into the details there are some design aspects that are common for all the following experiments. These were made according to findings in our previous work (mostly [Pav03]).

- **Training algorithm.** The incremental version of the backpropagation algorithm was chosen because its rate of convergence was considerably faster than other training strategies (e.g. RPROP, [Rie93])

- **Training vector shuffling.** In order for the incremental backpropagation to converge it is necessary to present the training vectors in random order. This may be a problem because the training data are so large that they do not fit into memory. This is solved by dividing the training data into smaller parts and the following change in the shuffling process: First, the list of input files (i.e. the parametrized recordings and their respective label files) is shuffled and divided into parts. Second, the individual training vectors in each part are shuffled.

- **Zero mean normalization.** Our previous work has shown that it is beneficial to normalize the feature vectors to have zero mean. This

was usually done by computing the mean of the whole recording and then subtracting it from each feature vector. Unfortunately this poses a problem when live recognition is performed since each speech frame is processed immediately by all the components of the recognition system and it is not possible to wait for the computation of the mean of the entire recording. The solution was to compute an estimate of the mean (from a set of recordings) and store it for the subtraction during live recognition. We have found that this estimate is different for each microphone/sound card setup an thus complicates live recognition by requiring a calibration step. We have also found out that if the neural network is sufficiently large (e.g. more than 1000 hidden units) it is possible to achieve comparable results without zero mean normalization which led to a decision to abandon zero mean normalization completely.

- **Three layers.** It has been stated in section 3.1.2 that three layers of neurons (two layers of weights) are sufficient for a MLP to approximate any function. We have not done experiments with only two layers but they are discussed in [Teb95] which shows that removing the hidden layer led to a serious degradation of the recognition accuracy.

7.1 MLP Layer Sizes

7.1.1 Input Layer

The size of the input layer is related to *context sensitivity*: it is possible to add features from the frames surrounding the one currently processed (see sections 4.4 and 5.4.1) allowing the network to "see" beyond the small speech segment that is being classified. The context window is increased by adding speech frames from each side of the currently recognized frame.

Increasing the size of the input layer also increases the number of trainable weights. In order to show that the increase in recognition accuracy is due to the size of the context window and not due to the increased number of weights we have run two sets of experiments: one where the size of the hidden layer was fixed to 1000 neurons (denoted "1000" in Figure 7.1) and one with a variable size of the hidden layer (denoted "variable"). In the second set the hidden layer size was chosen in such a way that the total number of weights was always the same.

It can be seen from the results on both corpora that the benefit of having a larger context as opposed to a single frame on input is enormous. A context window of nine consecutive frames was chosen as an optimal setting because the addition of further frames did not lead to an increase in accuracy.

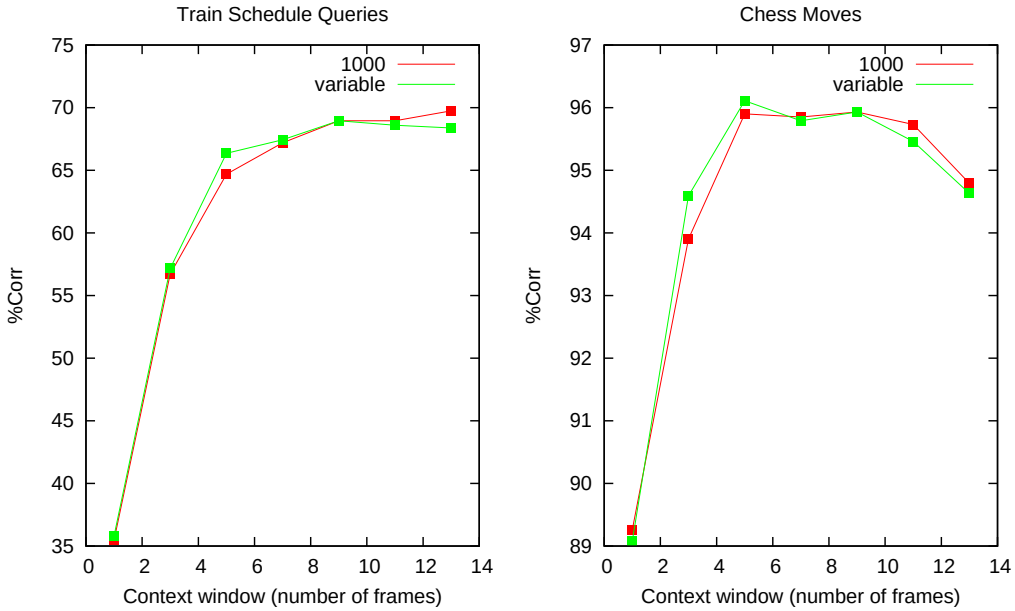

Figure 7.1: Relation of context window size and the percentage of correctly recognized words.

7.1.2 Hidden Layer

When the sizes of the input and output layers are decided it is possible to increase the recognition accuracy by increasing the number of hidden neurons. The price for adding hidden neurons is an increase in computational costs.

Hidden neurons	Weights	Chess Moves	Train Schedules	Numbers
1000	153000	95.90	74.97	95.75
2000	306000	96.89	77.21	94.76
4000	612000	96.84	78.32	93.31

Table 7.1: Percentage of correctly recognized words for different hidden layer sizes and different testing corpora.

Table 7.1 shows the results for different hidden layer sizes. It can be seen from the results for the Numbers corpus that the statement that more hidden neurons means higher accuracy is not universally true. Since the sentences in the Numbers corpus were not present in the training data we can hypothesize that overtraining occurs in this case. The higher number of trainable parameters allows the network to learn the variants of the phonemes

occurring in the training corpora while harming generalization at the same time.

7.1.3 Output Layer

The number of output neurons depends on the choice of phonetic units. An important question is whether it is advantageous to have separate state models for each monophone phonetic unit as is the case of GMM acoustic models or whether state duplication is sufficient. State duplication is discussed in section 8.2.3 from which it can be concluded that in most cases (except for the Numbers corpus) three states per monophone lead to the best results.

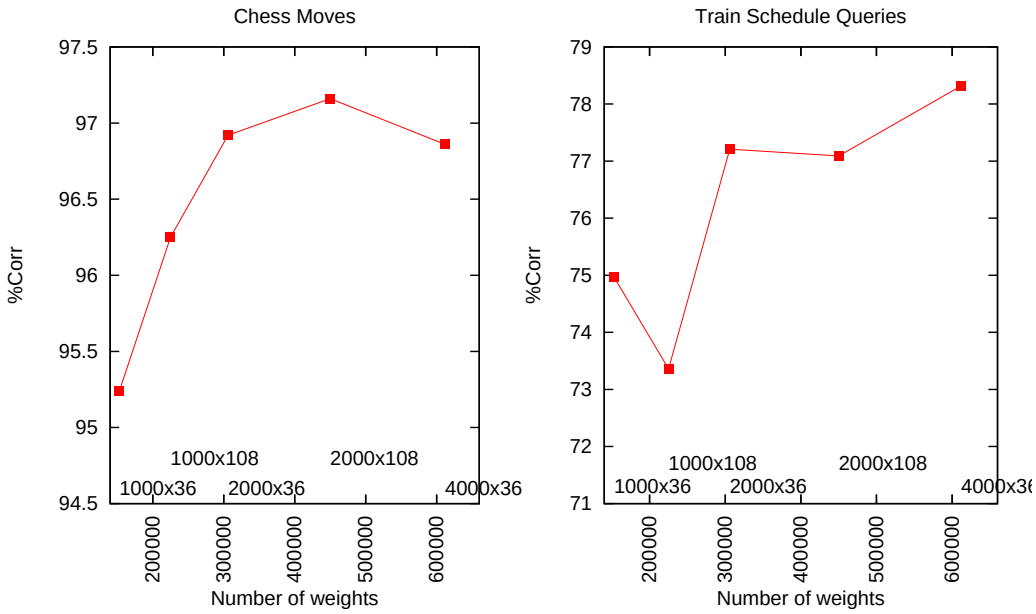

Figure 7.2: A comparison of different networks based on the number of trainable weights.

To test what impact on performance does having three separate state models for each monophone it must be taken into consideration that this increases the size of the output layer three times which results in larger number of weights. Figure 7.2 shows a comparison of different networks based on the number of weights (In the notation the first number represents the number of hidden neurons. The second is the number of output neurons, e.g. "1000x36"). We conclude that having separate state models can lead to some improvement in recognition accuracy but it may as well be due to the

increase in the number of weights. For this reason we have decided to use only state duplication in our further experiments.

This section discusses only monophone units; for details on context dependent units see section 7.5.

7.2 Training

The training of neural network acoustic models is much more expensive in terms of computation times than the training of GMM models. One of the reasons is that there is no fast parameter estimation algorithm, the weights of the neural network are adjusted by some sort of gradient descent optimization (see section 3.2).

As has been stated at the beginning of this chapter we use the incremental version of the backpropagation algorithm. The training consists of the following steps:

1. Having one training vector pair (i.e. one input vector and its respective desired output vector) the error gradients for all the weights are computed (see section 3.2.1).

2. The weights are adjusted according to equation 3.19.

3. The training data are divided into smaller parts (in order to fit into memory) and steps 1 and 2 are performed for each training vector pair in each part. After a part is processed in this way a validation is performed: testing data consisting of vectors created from the train schedules testing set (see section 5.3) is presented to the network that was partially trained by this training data part and mean square error is computed (see eq. 5.3). After all parts of the training data are processed we say that one *iteration* has been performed.

4. It is expected that after each iteration the mean square error computed on the validation data will decrease, but the rate of decrease will be lower with each iteration and after some amount of iteration the change in error becomes insignificant. We call this the convergence of the training algorithm.

Rather than stopping the training process after the decrease in error goes below a threshold we have decided to perform a fixed number of iterations. After observing several trainings the number was set to 30[1] and we are looking

[1]Our previous experiments have shown that the error decrease is only very slight with more than 30 iterations

for a setup that achieves the best results in these constraints. There are several factors that can affect the speed of convergence of the training process:

- **Learning rate.** The learning rate controls the proportion by which the weights are changed during the training process. If the rate is too low the validation error will converge too slowly. If it is too large the experiments show that this leads to a higher error in the end.

- **Activation function.** Both the literature [Teb95] and our previous experiments have shown that using a symmetric logistic activation function in the hidden layer instead of logistic may lead to faster convergence.

- **Error criterion.** The cross entropy error criterion (eq. 3.17) is sometimes [BM97] used as an alternative to the more common summed square error (eq. 3.6) and is said to lead to an improvement in convergence.

To see the effect of the above mentioned factors on the progression of the validation error as well as on the resulting word error rates we have tried various learning rates in combination with the different error criteria and activation functions. The experiment was carried out with our smallest network that had 1000 hidden neurons.

A larger number of learning rates was tested than is shown in Figure 7.3 but only the ones close to the best results are displayed. From these results we conclude that:

- Smaller learning rates need to be used when training with cross entropy error criterion. A too large learning rate may lead to very poor results as the word error for symmetric activation function with the learning rate of 0.0005 shows (the word accuracy at around 13% means that probably only silence was recognized correctly).

- The overall performance of the cross entropy error criterion is better than the performance of the summed square error. The evidence we present for this conclusion is the following: 1. The cross entropy training led to the best overall result (cross entropy + learning rate of 0.0001 + logistic activation function). But if we compare the best word result for cross entropy with the best result for summed squared error the difference may not be statistically significant. 2. Even though the ending MSEs for summed squared error vary less than in the case of cross entropy, the word error rates vary more for the summed squared error (if the extreme case for cross entropy is discarded).

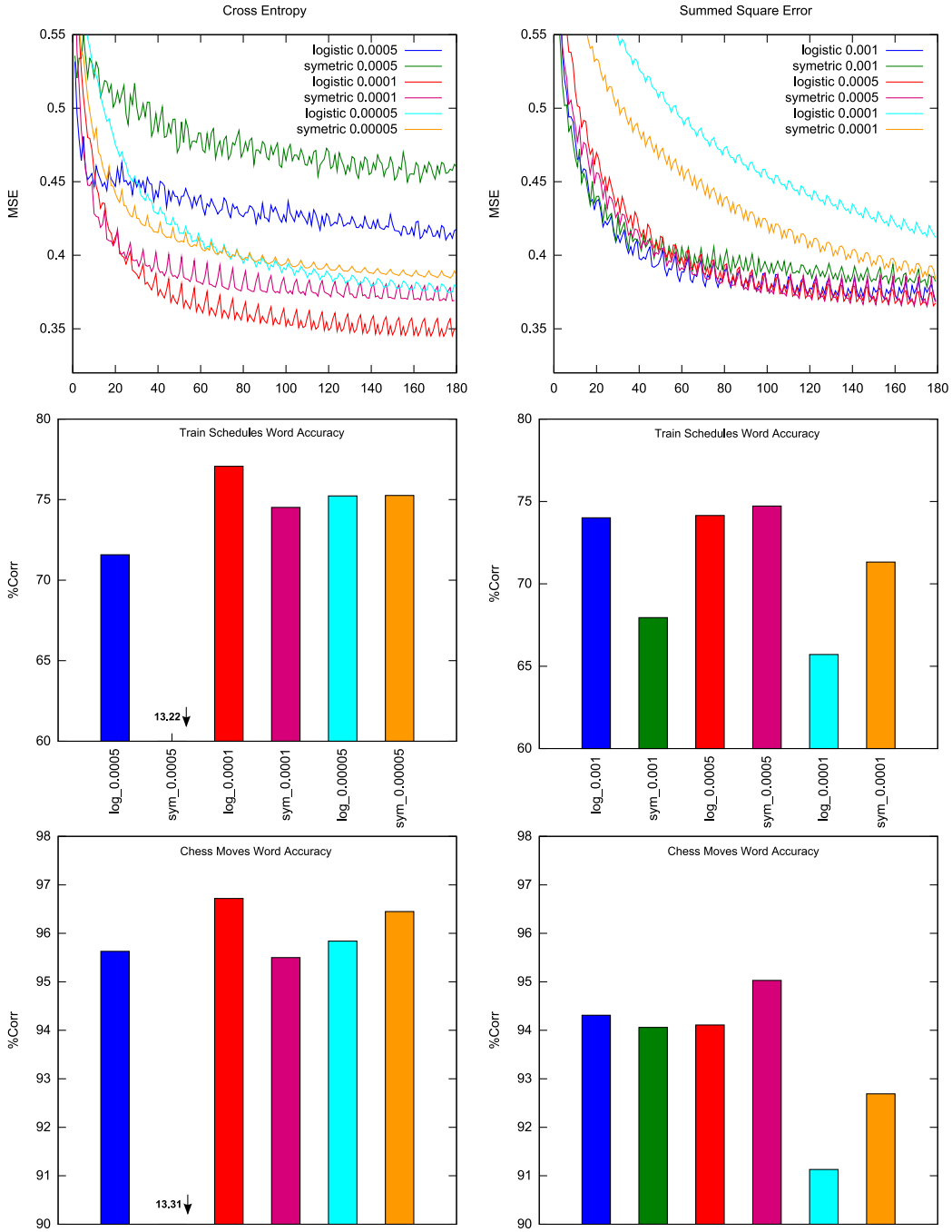

Figure 7.3: Results for different learning rates, error criteria and activation functions in hidden layer. The x axes of the upper-most graphs represent the number of training data parts presented to the network during training. To get the number of iterations this number has to be divided by the total number of training data parts, which in this case was 6.

- Despite initial assumptions the best results were achieved with logistic activation functions in the hidden layer. Again the difference for word error rates between symmetric and logistic activation function may not be significant, because the confidence intervals overlap. To test if this difference is coincidental a second test was made with a network that had 2000 hidden neurons. This led to very similar results as in the case with 1000 hidden neurons.

7.2.1 Adaptive Learning Rate

Both the literature [Teb95], [BM97] and our previous experiments show that it can be beneficial to adjust the learning rate during the course of training. The basic idea is to use higher learning rates at the beginning of the training process and decrease the learning rate over time. Two learning rate adjusting schemes were tested:

- **Adaptive learning rate.** In each iteration three neural networks are trained: one with the current learning rate, one with a double learning rate and one with half the current learning rate. After the iteration is finished a validation is performed and the network leading to the best results is used in the next iteration with its respective learning rate as the current rate.

- **Fixed set of learning rates**. First a set of learning rates is chosen that will be used during training. In our experiment the set was $\eta \in \{0.0005, 0.0002, 0.0001, 0.00005, 0.00002\}$. In each iteration there is a separate network for each rate in the set. Validation is performed after the training and the best network is used in the next iteration.

The advantage of the above described schemes is that all the networks in one iteration can be trained in parallel. Figure 7.4 shows the results for both learning rate schemes compared to training with a single learning rate (only the validation results from the best performing network in each validation are shown). The adaptive learning rate scheme converged to extremely small learning rates and did not give very satisfactory results at the end.

The results obtained with the fixed set of learning rates are comparable with the results from a single learning rate. The conclusion of this experiment is that neither scheme produced any benefits in comparison to using only one learning rate during the whole training process.

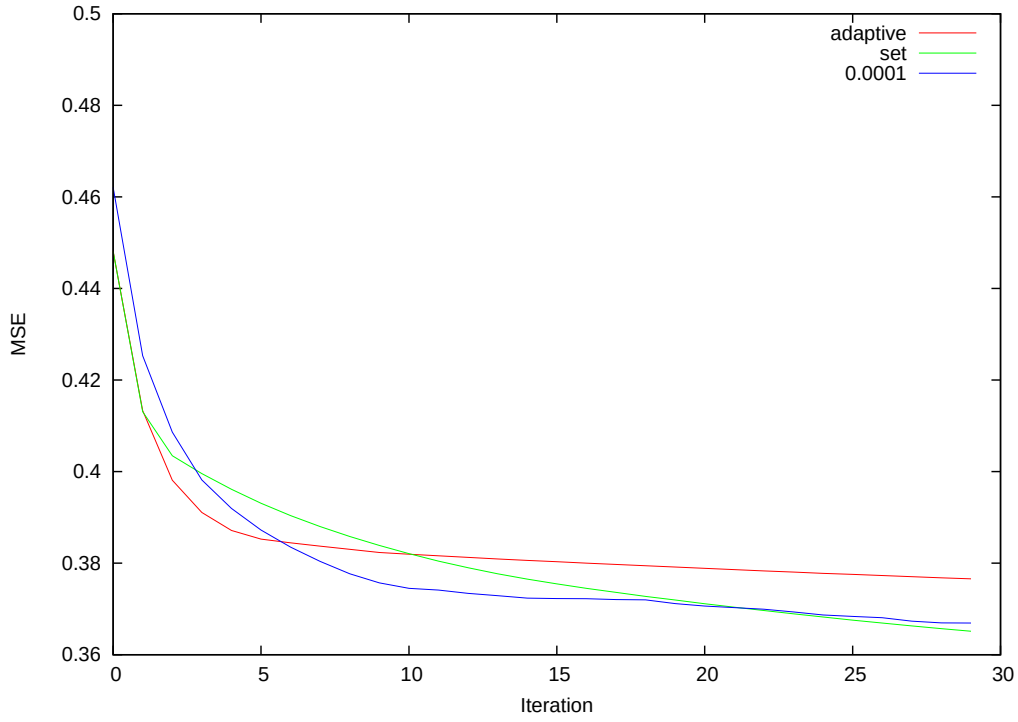

Figure 7.4: Training with learning rate that varies over time.

7.2.2 Training Duration

All the neural networks were trained on a multiprocessor machine with 8 dual core Intel® Xeon™ 3.0 GHz processors. For various reasons[2] it is not possible to give exact times but approximate training durations can be found in Table 7.2.

Network	Duration [hours]	Duration [days]
1000	50	2.1
2000	70	2.9
4000	141	5.9
1000x517	254	10.6
2000x517	526	21.9

Table 7.2: Training durations for various neural networks.

[2]When the training takes a longer time (e.g. weeks) one must take into consideration events such as power and connectivity outages. Also, the training was done in a multi user environment so it was not guaranteed that the training application had 100% available processor time during the whole course of the training.

The training was done in a multi user environment and it is assumed that a free processor was available for each training process. However this cannot be guaranteed. Also when the networks for context dependent phonetic units were trained (1000x517 and 2000x517) the training process was usually at least once interrupted by a power outage or network maintenance, so the training durations for these are projections from the longest uninterrupted runs.

7.3 Phonetic Unit Priors

As has been stated in section 3.1.2 the outputs of a trained multi layer perceptron ca be interpreted as class posterior probabilities $P(class|input)$. But hidden Markov models work with likelihoods $P(input|class)$. These can be converted using the Bayes theorem:

$$P(input|class) = \frac{P(class|input) \cdot P(input)}{P(class)}. \tag{7.1}$$

Since the probability of an input $P(input)$ is the same for all HMM states examined in a given frame it can be discarded from the equation without affecting the result found by the Viterbi search. In order to make the conversion the outputs from a neural network need to be divided by the phonetic unit priors (in case of phonetic units composed of multiple states by state priors).

Figure 7.5 shows how the division by phonetic unit priors affect the recognition accuracy. The prior probabilities were computed as relative frequencies from the training data for the neural network (the frames were labeled, i.e. it was known which phonetic unit does each frame belong to).

When tested on the train schedule corpus the division by priors significantly (except for the case of the context dependent units, where the confidence intervals overlap) improved recognition accuracy. In the case of chess moves the introduction of information about prior probabilities was actually harmful to recognition accuracy, especially in the case of context dependent units.

We have suspected that since the chess moves corpus is very specific the composition of words could be quite different from the training corpus. The chess moves corpus contains a small set of words that occur with much higher frequencies: chess piece names, row numbers and letters representing columns. For this reason there might be a mismatch between phonetic unit frequencies computed on training data and frequencies in the chess moves corpus. To test whether this is the cause of the accuracy drop connected

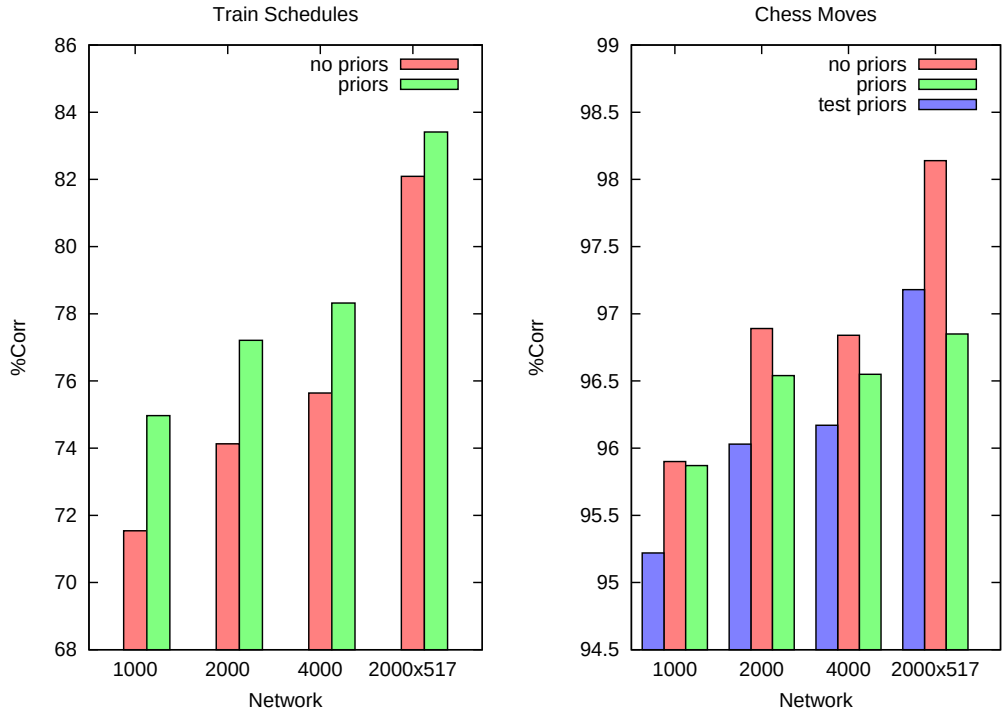

Figure 7.5: Neural network acoustic model with and without phonetic unit priors.

with the division by priors we have computed a new set of priors on the test (chess moves) corpus. It can be seen in Figure 7.5 that this did not lead to improvement in accuracy.

7.4 Alignment

When a neural network is trained as a 1-of-N classifier it is necessary to know to which class each training vector belongs to, i.e. the vectors need to be labeled. This may be done by hand [Pav03] but it is a very tedious and time consuming process. A more common approach is to generate the labels automatically by employing a trained speech recognizer (e.g. GMM based one, since the GMM parameters can be estimated by embedded training, see section 2.3.2). We have tested two approaches for automatic label generation:

- **Forced Viterbi Alignment.** An HMM is constructed by concatenating all the phonetic unit models based on the phonetic transcription of the utterance that is being labeled. After a Viterbi algorithm is run with this HMM to perform time alignment of its states, i.e. to find out

which state is most likely to have generated a given speech frame. The result is then used to construct the desired output vector by setting the desired activation of the neuron corresponding with the label to one and setting all other desired activations to zero.

- **Soft Alignment.** The previous method makes "hard" decisions, i.e. the state either generated the speech frame or it did not. It is also possible to give probabilistic "soft" results. A HMM is constructed in the same way as in the previous case but instead of the Viterbi algorithm the forward and backward variables described in section 2.2.3 are computed. Having these the probability of being in a given state at a given time can be computed (See eq. 2.34). The probability of a phonetic unit can be computed by summing the probabilities for all the states in the given frame that belong to the phonetic unit. The desired output vector is then constructed from these probabilities.

Once a neural network is trained with the automatically labeled training data it can be used to generate a new set of labels. This is referred to as *recursive labeling*. It has been reported e.g. by [Teb95]) that recursive labeling can further improve recognition accuracy if a new network is trained with the labels generated by the first network.

Train Schedules				
Alignment	Forced Viterbi		Soft	
Network	GMM	Recursive	GMM	Recursive
1000	74.97	74.52	74.75	75.03
2000	77.21	77.54	76.01	78.07
2000x517	N/A	83.15	83.41	83.31
Chess Moves				
Alignment	Forced Viterbi		Soft	
Network	GMM	Recursive	GMM	Recursive
1000	95.90	95.50	96.35	95.81
2000	96.89	96.95	96.27	96.81
2000x517	N/A	98.07	98.14	97.72

Table 7.3: A comparison of different methods for automatic label generation.

Table 7.3 shows the values of $\%Corr$ we have obtained be employing various methods for automatic alignment of labels. The last line in each Table (2000x517) shows the results for context dependent phonetic units (see section 7.5). The first sets of labels were generated by a 32 mixture GMM. The second generation of labels (denoted Recursive in the Table) for context

independent phonetic units was provided by the trained network with 2000 hidden neurons (namely the one trained on labels generated by forced Viterbi alignment). The same was done for context dependent phonetic units.

From the results presented in the table we conclude that it is sufficient to train with labels generated by forced Viterbi alignment using Gaussian mixture acoustic models. Given the size of our testing data (see appendix C for confidence intervals) there is no observable improvement when soft alignment is performed or when recursive labeling is done.

7.5 Context Dependent Phonetic Units

Context dependent neural network acoustic models suffer from the same problems as their GMM counterparts. The larger number of phonetic units means that more trainable parameters are necessary to properly model their probability distributions. This means rising computational costs (especially during the training phase) and the sparse data problem since some phonetic units may be undertrained.

Some authors [Bou92] try to solve the problems by computing separately the context probabilities based on the input o_t and a precomputed monophone probability $P(S_j|o_t)$. If there are L context classes $C = \{c_1, \ldots, c_L\}$ then the probability of the context dependent unit can be expressed as

$$P(S_j, c_k|o_t) = P(S_j|o_t)P(c_k|S_j, o_t). \tag{7.2}$$

The term $P(S_j|o_t)$ can be computed by a monophone neural network such as those discussed in the previous sections. The context probability $P(c_k|S_j, o_t)$ can also be computed by a neural network, but the problem is that the outputs of such a network must be computed separately for each context independent phonetic unit S_j. The authors present a non-trivial workaround but the paper only discusses preliminary results so it is not clear how well this approach works.

We have tested a much simpler solution which takes advantage of decision tree clustering done for the GMM acoustic models. If we use the most strict threshold from Table 6.1 then the total number of leaves of all the decision trees is only 517. Since the decision tree provides mapping from any arbitrary triphone to a model represented by the leaf the leaves can be treated in the same way as regular context independent phonetic units.

The resulting network has the same inputs as a monophone network and 517 output units. A trained GMM model can be used to generate class labels for the training data via forced Viterbi alignment.

The first network that we trained had 1000 hidden units and as can be seen from Table 7.4 the resulting recognition accuracy was rather poor. We suspect that because neural networks model boundaries among classes and the number of classes was increased in comparison with the 36 monophone models the number of hidden neurons was not sufficient to properly model those boundaries.

The interesting results came when the number of hidden neurons was increased to 2000. It can be seen that for the chess moves corpus the network outperforms any other model that we have tried. In the case of the train schedule corpus the results are better in absolute numbers, but the confidence intervals overlap so the difference may not be statistically significant.

The introduction of triphone phonetic units brought a considerable increase in the number of weights the network has. To test whether the increase in recognition accuracy was due to context dependency rather than due to the increase in the number of trainable parameters (weights) we have trained a monophone network with 8000 hidden neurons which has similar amount of weights to the triphone network with 2000 hidden units. The results in Table 7.4 show that the accuracy achieved with this network is much lower. We conclude that context dependency in neural network acoustic models has notable benefits.

The last line of the Table gives the results achieved with the triphone GMM model that was used to generate class labels for the training data.

	Chess Moves		**Train Schedules**	
Acoustic model	%Corr	Acc	%Corr	Acc
net1000x517	92.83	92.81	71.73	69.46
net2000x517	98.14	98.14	83.44	80.89
net8000	96.56	96.55	77.85	74.32
triphone GMM	97.52	97.23	82.56	80.05

Table 7.4: Test results for context dependent phonetic units

Chapter 8

Decoding

This chapter describes the issues encountered during the design of a decoder that works with both variants of acoustic models discussed in this work. We have decided that in order to compare the neural and GMM acoustic models we need to tune the parameters of the search algorithms separately for each of the acoustic model variants and and compare the best performing setups. Other aspects of the decoder design are mentioned as well, mainly those that affect the speed of the decoding process.

There are several characteristics that are common for all the versions of the decoder discussed in this chapter, namely:

- The decoder is one-pass, i.e. the final result is known after one run of the search algorithm.
- Only one sequence of words is returned, there is no N-best list generation.
- The decoder is time synchronous, the speech frames are processed one at a time.

8.1 Relation to Acoustic Models

When evaluating the performance of acoustic models one usually thinks in terms of recognition speed and recognition accuracy[1]. Both can be significantly affected by the setup of the decoding process.

The search is usually pruned and this means that while a single speech frame is processed the number of phonetic unit probabilities that are requested (this is especially true for context dependent phonetic units, see

[1]Regardless of whether we are talking about $\%Corr$ or Acc, see section 5.1.1.

Table 8.1) can be smaller than the total number of phonetic units. To take advantage of this fact conditional computation of acoustic model probabilities is introduced: only those probabilities that are requested during the search are computed.

While the conditional computation can be easily implemented for the GMM models (probabilities for each phonetic unit are computed separately, see equation 2.8) it cannot be done for the neural network model where the parameters (i.e. weights) are shared among phonetic units. We have solved this by:

1. Computing the activations for all neurons in the hidden layer.

2. Computing the activations for only those neurons in the output layer that correspond to those phonetic units that are requested by the decoder.

Even though the hidden layer activations need to be computed for all frames this still speeds up the computation. In the case of our network for context dependent phonetic units (described in section 7.5) the weights in the output layer take more than 80% of the total number of weights in the network.

The recognition accuracy can also be affected during decoding, see state duplication (section 8.2.3) and word insertion penalty (section 8.4.3).

		% of acoustic model computations	
acoustic model	no. of units	Chess Moves	Train Schedules
net1000	36	66.65%	96.43%
GMM 32 mix	108	43.67%	81.13%
net2000x517	517	14.94%	46.72%

Table 8.1: Percentage of phonetic unit probability computations for various acoustic models.

8.2 Recognition Graph

The goal in the designing of a decoder is to efficiently implement the Viterbi algorithm. A proper organization of the elements of an HMM (see section 2.1.1) is necessary to reach feasible computational and memory demands. The recognition graph is an oriented graph representing the HMM. The graph links serve as a sparse implementation of the transition matrix of the HMM since only the non zero transition probabilities need to be stored.

The HMM represented by the graph is not directly trained by the methods described in section 2.2.3. The graph is constructed by gluing together smaller HMMs representing the phonetic units (see Figure 8.1). Each phonetic unit HMM has several states which can either share the same acoustic model (see state duplication, sec. 8.2.3) or have separate acoustic models for each state.

The graph can be constructed from a pronunciation dictionary (see section 8.2.4 for discussion) of from a hand written grammar (e.g. an EBNF grammar). If there is no recursion in the grammar rules it can be converted to an oriented graph representing word transitions. After that the words only need to be replaced by their HMM representations.

Each node of the graph holds the following information:

- The acoustic model that is associated with this node. This can be an output neuron or a Gaussian Mixture. With this information the emission probability $b_j(o)$ can be computed.
- Whether the node is emitting (i.e. whether there exists a corresponding acoustic model, see null nodes, sec. 8.2.2).
- If this node is a word-end node, it contains information about the word (i.e. the orthographic transcription of the word).
- A list of the node's immediate successors (a list of links from this node). This usually includes link to self.
- An optional list of transition probabilities for all the links from this node.
- An optional reference to the decoding cell in the current active list (see section 8.3.1).

8.2.1 Transition Probabilities

When working with GMM acoustic models the recognition graph is constructed from small (usually three state) phonetic unit HMMs. These HMMs are always left to right (see section 2.1.3) and have transition probabilities estimated by the training algorithm. In HTK [You02] these are glued together in non-emitting entry and exit states that are present at the beginning and at the end of the phonetic unit HMM (see Figure 8.1 and section 2.3.2).

The transition probability (a_{01} in the Figure) from the entry state always equals 1 and so the transition probability between in-word phonetic unit HMMs equals the transition probability from the last state (no. 3 in the

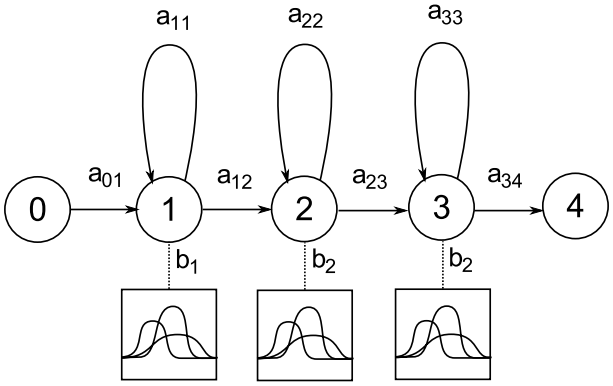

Figure 8.1: Left-right HMM of a phonetic unit with non-emitting entry and exit states and GMM acoustic models.

Figure) to the exit state (i.e. the transition probability a_{34}). For this reason the phonetic unit HMMs can be glued directly and the non-emitting states discarded. In our recognition system this is the preferred way and non-emitting states are reserved for special purposes (see section 8.2.2).

The transition probabilities between word models can be problematic because the probabilities for all the links exiting a word-end state usually do not sum to one. This can be solved by introducing a word transition penalty during the decoding process (see section 8.4.3).

In the case of neural network acoustic models only the emission probability models are trained which leaves an open question about what to do with the transition probabilities. Most proponents of hybrid HMM systems (e.g. [Teb95]) claim that:

1. The exponential distribution of state duration implied by the model is not appropriate for speech recognition where the duration of phonemes has a bell shaped distribution.

2. Duration modeling plays a relatively small role in recognition accuracy.

For this reason most of the hybrid systems (including ours) assume uniform distribution of all transition probabilities. We have decided to test the plausibility of the usage of uniform distribution of transition probabilities by testing both trained and uniform transition probabilities. Table 8.2 shows the results. It can be seen that while the change of the transition probability distribution probably did not affect the performance on the Chess Moves corpus, the accuracy for the Train Schedule corpus dropped significantly. Even though we did not try transition probability modeling for neural network acoustic models, these results present an opportunity for future research.

	Transition Probabilities	
Corpus	Trained	Uniform
Chess moves	95.90	96.04
Train schedules	71.46	70.03

Table 8.2: Percentage of correctly recognized words for different transition probabilities. The HMM had GMM based acoustic model with 32 mixtures.

8.2.2 Null Nodes

It is often the case that a subgraph similar to the one depicted in the left side of Figure 8.2 is contained in the recognition graph. This happens when transitions from end nodes from a group of words is allowed to all the beginning states in another group of words. If the number of words in the first group is N and the number of words in the second is M then the total number of links between these groups is $N \times M$. Each of these links adds computation during the decoding phase.

If the probabilities of links coming from the end of each single word in the first group are equal (this is not true if the probabilities of the links between the words were taken from e.g. bigram language model) then a non-emitting null node can be inserted. This reduces the total number of links between the two groups to $N + M$. Null node can e.g. serve as the root node of a tree organized lexicon.

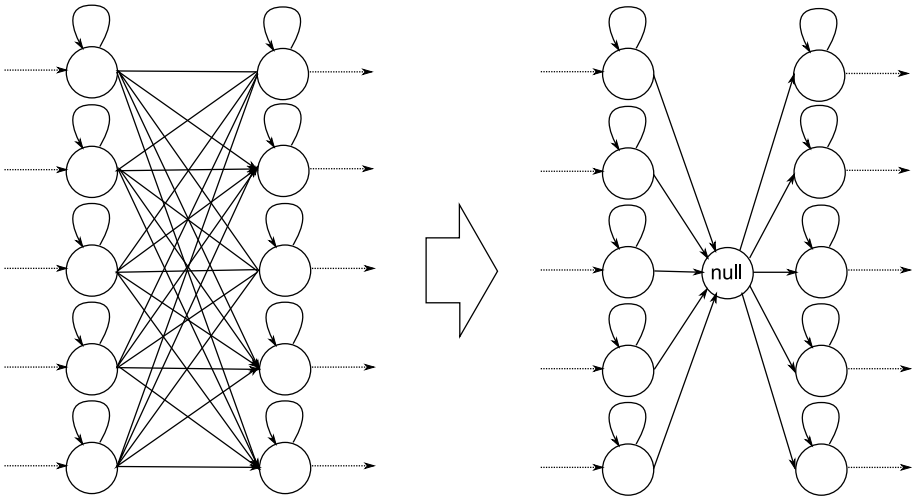

Figure 8.2: Insertion of a null node decreases the total number of transitions.

The non emitting nodes are modeled as HMM states that emit nothing at the time "between" speech frames. This can be handled during the decoding

phase by ensuring that the null nodes for each frame are processed only after all emitting nodes for the frame were processed. This can be done by sorting the indexes of the recognition graph or by keeping separate active lists for emitting and non emitting nodes.

8.2.3 State Duplication

When working with a neural network acoustic model many incorrectly recognized word sequences contain words that are too short to have been pronounced by a real speaker. Those are more likely due to errors produced by the acoustic model. Such errors could be avoided by having some constraints on the minimal duration of HMM states.

Our first attempt was to count the number of frames for each cell in the Viterbi decoding algorithm. Transition to another state was only allowed after the number of observations (speech frames) that the state generated was greater or equal the minimum number. Unfortunately the resulting recognition accuracy was very poor and this method was abandoned immediately.

The reason why the method failed can be found in [Teb95] and is demonstrated in left side of Figure 8.3. Suppose the HMM subgraph for the word "ano" (means "yes" in Czech) is a part of a larger recognition graph and the minimum duration is two frames. The two paths marked red and blue both represent valid paths since each path stays in one state for at least the time equal to the length of two speech frames. The problem can be seen at the place marked by the green circle: If the red path has higher score at this point, it will overwrite the search variables for the blue path even if the total score of the blue path is higher than the one of the red path. This makes the search inadmissible since maxima at local level may prevent global maxima to be found.

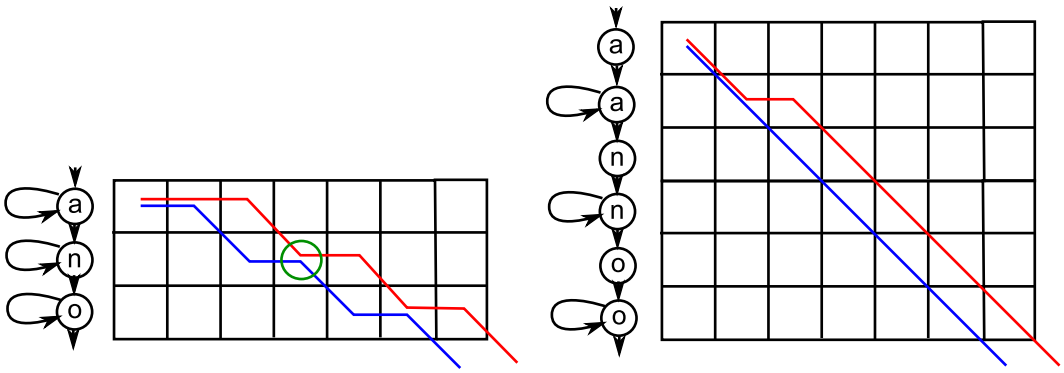

Figure 8.3: Frame count and state duplication as duration constraints.

State duplication overcomes this problem as can be seen from the right side of Figure 8.3. An important question is, if higher recognition accuracy is achieved by doubling (or tripling for three states per phonetic unit) the size of the whole recognition graph, how does this affect the recognition speed?

Table 8.3 shows that as the number of states per phonetic unit increases the average number of active states actually decreases and so does the recognition speed. At certain point the duration constraint is too high to reflect the true durations in the recognized speech data (i.e. the speakers speak faster than the model assumes). Note that for the Numbers corpus this happens with just three states per phonetic unit. An interesting side effect is that we can adapt to faster speaking rates by just changing the number of states without having to adjust the sampling rate for feature extraction (which would require to retrain the acoustic models).

Train Schedules				
no. of states	$\%Corr$	no. of act. states	% of act. states	$\%RT$
1	67.40	2017	46.82%	6.17%
2	71.15	1142	13.26%	5.08%
3	71.62	726	5.62%	4.49%
4	58.02	566	3.28%	4.11%
Chess Moves				
no. of states	$\%Corr$	no. of act. states	% of act. states	$\%RT$
1	94.42	259	34.52%	3.47%
2	95.47	238	15.91%	3.49%
3	95.87	202	8.99%	3.44%
4	95.02	187	6.24%	3.34%
Numbers				
no. of states	$\%Corr$	no. of act. states	% of act. states	$\%RT$
1	94.96	1008	45.31%	4.80%
2	95.75	975	21.93%	4.36%
3	90.99	906	13.57%	4.13%
4	72.73	977	10.98%	4.20%

Table 8.3: Performance of a recognition system for different number of states per phonetic unit.

8.2.4 Tree Organized Lexicon

Suppose we need to perform continuous speech recognition and there are no language models (language models are discussed in section 8.5). Transition

from each word to any other word will be allowed and the total number of words is N. The word models can be constructed by gluing together phonetic unit HMMs according to a pronunciation dictionary. If the recognition graph was implemented in the most straight forward way there would be N^2 inter-word links. Such a large number of links would considerably increase the computational costs so a null node is inserted as seen if the left side of Figure 8.4. From now on this kind of graph will be referred to as *linear lexicon*.

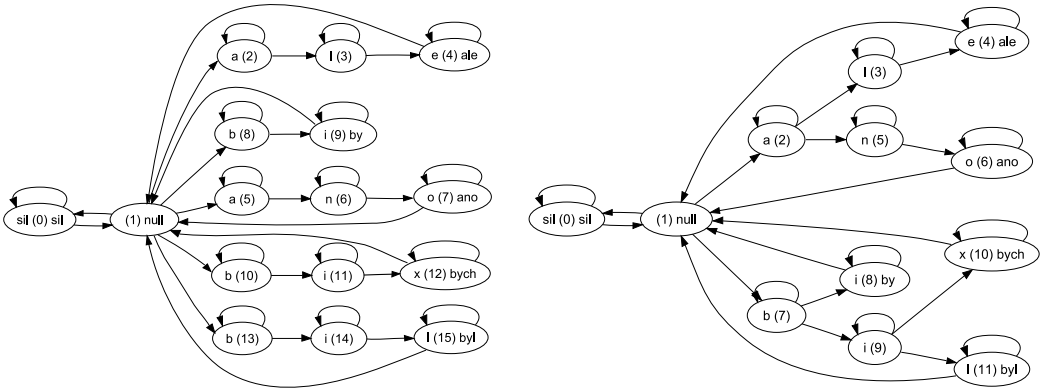

Figure 8.4: Linear and tree organized lexicon.

If a pruned search is performed with such a graph it can be observed that:

- Most of computations are performed for states at the beginnings of words where there is usually not enough information to prune. As an example consider multiple word beginnings in one time frame: their respective scores differ only in the emission probabilities of the states at the beginnings of these words. As the search moves closer to the word ends the difference between scores rises (if we compare scores of two paths through the graph their differences rise with the length of the path) and this allows for pruning of a larger number of states.

- The computations performed for some states (e.g. states 8, 10 and 13 in Figure 8.4) lead to the same results, since these states have the same emission probability.

These facts can be exploited by organizing the graph as a prefix tree (see right side of Figure 8.4). This graph will further be referred as *tree organized lexicon*. This allows for shared computations for a large number of word beginnings. Table 8.4 shows how tree organized lexicon can speed up the search. If no pruning is done the speed is increased only about three times but if pruned search is performed the speed increases by an order of magnitude.

graph	avg. no. of active states	%RT	relative speed improvement
linear no pruning	29040	103.31%	1.00×
tree no pruning	11624	34.29%	3.01×
linear beam search	6681	33.44%	3.09×
tree beam search	726	7.67%	13.48×

Table 8.4: Linear vs. tree organization of lexicon, tested on the Train Schedules corpus

8.3 Search Organization

The previous section described how to store the elements of an HMM so that it can be efficiently searched. This section will deal with storage of the search variables of the Viterbi algorithm, i.e. mainly the variables δ (the score of the best path) and ψ (the backpointer) described in section 2.2.2.

The storage of the search variables can be viewed as a matrix (which may be sparse if pruning is done) of *cells* where each cell holds the search variables for the given HMM state at the given discrete time instant. The columns of the matrix correspond to speech frames and the rows correspond to nodes in the recognition graph. Each cell holds the following information:

1. The corresponding node of the recognition graph.
2. The score of the best path leading to this cell (i.e. the δ variable).
3. The backpointer to the previous cell in the best path (i.e. the ψ variable.
4. The corresponding time frame.
5. The identifier of this cell (this is only used in search with higher order stochastic language models otherwise it is equal to the number of the node of the recognition graph).

Note that if the variables really were stored in the matrix only the two main variables (δ and ψ) would be needed. Other ways of storage will be suggested in the rest of this section.

8.3.1 Active Lists

An *active list* is a data structure that holds the cells with search variables and provides the following operations:

1. Iteration through the active list.

2. Inserting a cell into the active list. If the cell with the given identifier is not in the active list it is added to the active list. If the cell is already in the active list it is replaced by the new one only if the new cell has a higher score.

3. Pruning of the active list (based on beam search or sorting, see sections 4.6.3, 8.4.1 and 8.4.2)

Since all the paths leading to the cells in an active list can be traced via the backpointers it is only necessary to keep the active list for the frame being currently processed.

The three different implementations of active lists that we have tested are listed below and their respective strengths and weaknesses are discussed.

Array Active List The most straightforward way is to store the search variable cells in an array. The identifier of the cell is used as an index into this array which guarantees a high speed of the insertion operation (since the check whether the cell already exists in the active list is fast).

The weak point of this implementation becomes obvious when pruned search is performed: the array must be large enough to contain the cells for all the nodes in the recognition graph. No matter what is the pruning threshold the whole array must be searched at least twice: first to find the highest score and second to find which states to expand. This can make the pruning operation quite expensive since the number of active states can be much lower than the total number of states in the recognition graph. In our experiments the decoding speed never increased more than two fold when pruning was performed together with array active list.

Hash Map Active List A typical hash map implementation provides both iteration and random access based on a numerical key which are needed to implement the operations on active lists. The advantage here is that the key range (i.e. the total number of states in the recognition graph) can be much larger than the number of cells stored in the active list. If the search is pruned, only the cells in the active list need to be examined when searching for the highest score and discarding cells during pruning actually reduces the number of states through which it is later necessary to iterate.

When higher order stochastic language models are used (see section 8.5) the recognition graph can be quite large since states with different word histories must be considered distinct. In such case it is inconvenient to keep

the whole recognition graph in the memory and instead generate it on-the-fly. This is done by keeping a graph without the language model in the memory and changing the key (the cell identifier) into the hash map. The key is then composed of two parts: the word history (length is based on the order of the language model) and the number node in the recognition graph.

In [Aub02] the decoding methods are divided into two classes: those with static or those with dynamic network expansion. The network there is our recognition graph. Static network means that the entire graph is in the memory while dynamic means that a portion of the graph is generated during the search. Storing the active list in the hash map allows dynamic network expansion; all the other methods described here allow only static network expansion.

Active List with Graph Storage The main disadvantage of hash map storage is the retrieval of a cell from an active list based on its identifier, since a hash needs to be computed from the identifier. The computation of the hash function is much more expensive than a simple array lookup. We are looking for an implementation that would allow both fast insertion and deletion as well as a fast way to find whether the cell for the current node's successor already exists.

This is achieved by storing the references to the cell in an active list in two places. One is a simple linked list which provides fast insertion[2] and deletion operations. The second place that stores the reference to the cell is the node of the recognition graph. This way, when examining all the nodes in the list of the current node's successors, it can be determined whether the cell belonging to the successor is already in the new active list: If the time frame of the cell in the graph is correct, the cell is present in the new active list. If the time information is out of date, the cell can be replaced by a new one.

Note that this implementation is very similar to the token passing algorithm described in [You89].

8.3.2 The Search Algorithm

The algorithm processes all the cells in the given frame and creates a new active list. The old active list is then discarded. The main part of the algorithm is a so called *node expansion* which searches all the successors of a node belonging to the cell being processed. This step replaces the original maximum search of the Viterbi algorithm.

[2]Insertion at the end of the list is sufficient, that is why it can be considered fast.

Algorithm 1 The search for the most likely state sequence.

for $t = 1$ to T **do**
  for all cells i in the active list **do**
    create an empty new active list and empty active list of null nodes
    prune the current active list
    expand all the nodes in the current active list
    expand all the nodes in the active list of null nodes
    current active list = new active list
  end for
end for

Algorithm 2 The node expansion procedure.

for all successors j of the expanded node i **do**
  create a new cell corresponding to node j
  if j is emitting **then**
    set the score $\delta_j(t+1) = \delta_i(t)a_{ij}b_j(o_{t+1})$
    set the backpointer $\psi = i$
    insert this cell into the new active list
  else
    set the score $\delta_j(t+1) = \delta_i(t)a_{ij}$
    set the backpointer $\psi = i$
    insert this cell into the active list of null nodes
  end if
end for

Once all the frames have been processed it is possible find the best scoring cell and trace back the best path via the backpointers. Since each cell holds information about it's corresponding graph node and some nodes (namely the word-end nodes) hold information about the word they represent it is easy to obtain the recognized word sequence in this way.

8.4 Tuning

The previous sections described how to implement the Viterbi algorithm in order to reach maximum recognition speed. The design aspects described so far do not affect the admissibility of the search, i.e. the decoder described in this chapter guarantees to find the single most likely state path just as well as the Viterbi algorithm described in section 2.2.2 does.

This section will discuss methods that do not guarantee search admissibility but have been experimentally shown to either increase recognition accuracy or recognition speed. This may be done by directly altering the scores during the search (the word transition penalty) or by not expanding those states that are believed not to be likely to affect the final solution (pruning).

8.4.1 Beam Search

Beam search is a non admissible path finding algorithm that saves computational and memory resources by only expanding a portion of all the possible search states. The number of states that are expanded is sometimes referred to as *beam width*. An important question is how to choose between states to be expanded and states to be pruned. The most common way is to use variable beam width. First the best scoring state is found:

$$\delta_{\max}(t) = \max_{\forall j} \delta_j(t) \tag{8.1}$$

Afterwards a minimum score $\delta_{\min}(t)$ is chosen as a portion of the maximum score determined by a threshold θ.

$$\delta_{\min}(t) = \theta \cdot \delta_{\max}(t), \qquad 0 < \theta < 1 \tag{8.2}$$

Only the states having score higher than $\delta_{\min}(t)$ are expanded. The threshold θ must be determined experimentally. During our experiments we have found that when searching for an optimal threshold it is better if the points at which the thresholds are tested have a logarithmic scale. For this reason the threshold is specified in a different form:

$$\theta = 10^{-\alpha} \tag{8.3}$$

where α is the input value of the threshold.

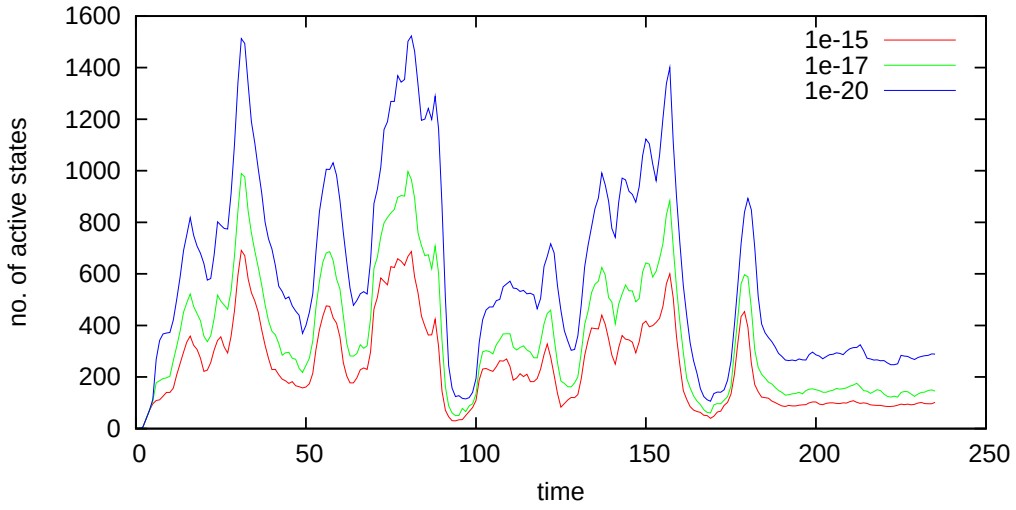

Figure 8.5: Number of active states during recognition of one utterance from the train schedule corpus.

Having the beam width corresponding to a proportion of a maximum score allows to efficiently deal with uncertainty: If there is a large number of states with similar scores, the beam is relatively wide. On the other hand if there is a small number of high scoring states (i.e. there is more certainty about the result) the beam width can be narrower resulting in lower computational expense. Figure 8.5 show the development of active states during the search with different thresholds.

Experiments show that only a small fraction of the states of the recognition graph needs to be active during the search. In the case of our testing corpora we have found that using a threshold that leads to an approximate average of 200 active states leads to the same results as if the search was not pruned. The 200 states is 1.6% of the total number of states (12387) in the HMM graph for the train schedule corpus and 9% of the total number of states (2211) in the graph for chess moves corpus.

To test how our recognition system responds to different beam thresholds we have run tests with both monophone and triphone models. The results can be seen in Figure 8.6. A typical recognition accuracy response starts low as a too strict threshold leads to errors during the search. With increasing beam width the accuracy rises steeply and eventually stabilizes. At this point the increase in beam width only means an increase in computational costs but not an increase in accuracy.

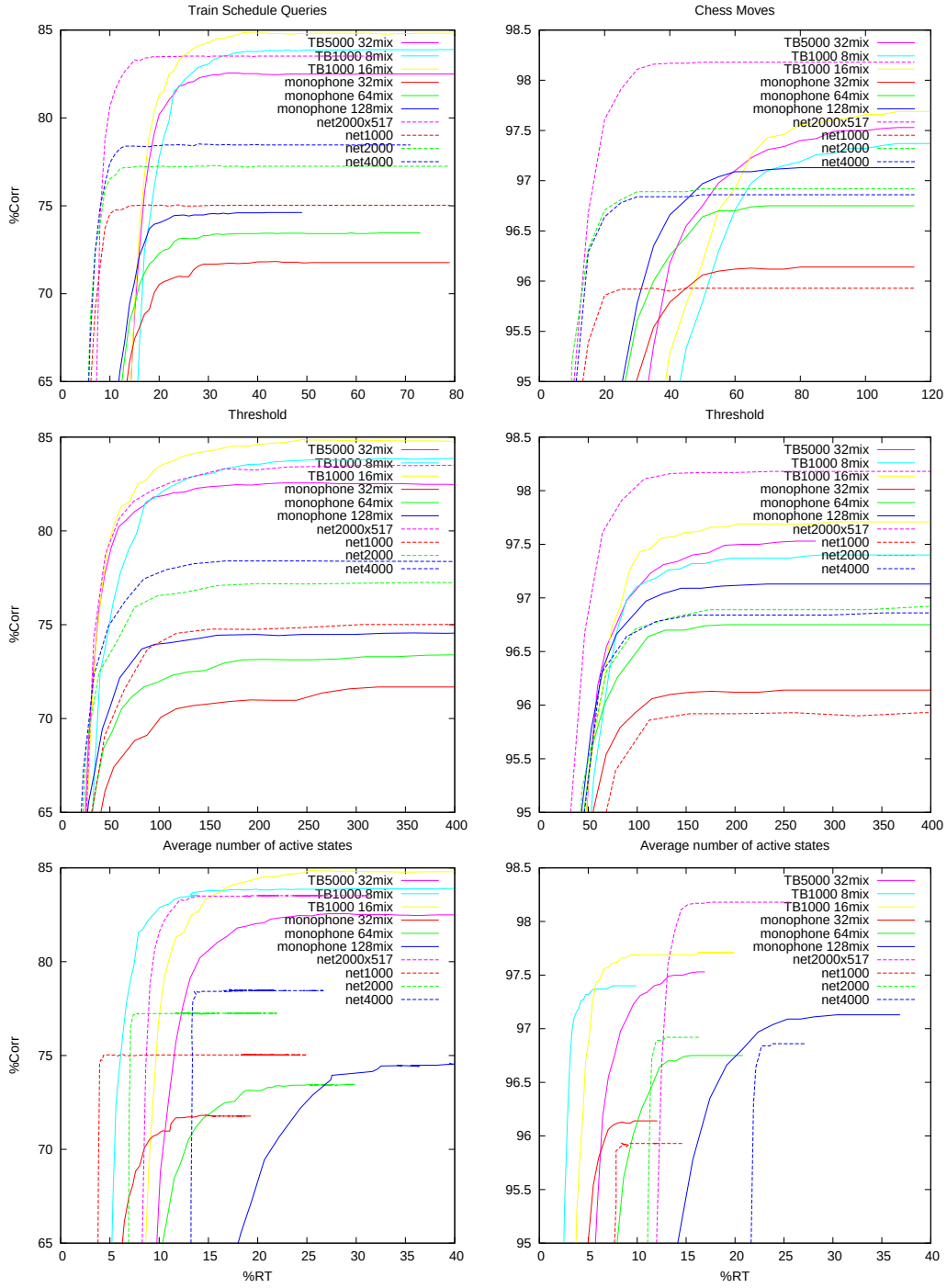

Figure 8.6: The accuracy response to the setting of the beam threshold.

The first row of Figure 8.6 shows how different models respond to different thresholds α. Generally, higher thresholds are needed for the chess moves corpus. Neural network based models need lower thresholds than GMM based ones (this is due to different dynamic ranges of probability outputs of these models).

The graphs in the second row of the Figure were created by computing the average numbers of active states during the whole test and comparing it with achieved recognition accuracy. The development of the accuracy response to an average number of active states during the search is similar across different models and corpora. When the number of active states per frame reaches 200 it can be said that the increase in accuracy has stopped for majority of models and, even though some models may reach slightly higher accuracy with more active states, the relative standing (in terms of accuracy) of all the models does not change. For this reason we have chosen 200 as the number of average active states that all our representative models (see section 9.1) should lead to (by setting the appropriate pruning threshold).

The graphs in the last row show the importance of beam threshold setting during the comparison of performance of different acoustic models. Since the number of active states during the search is limited there may be probabilities of phonetic units (in fact a large number of them) that do not need to be computed. For this reason pruning also saves computational resources during the computation of acoustic model probabilities (see section 8.1).

When comparing acoustic models one is usually interested in comparing the achieved accuracy with the recognition speed. The last row of Figure 8.6 shows that such performance is not fixed at a certain point in the accuracy/speed plane. Rather than that there is some room for choosing a balance between accuracy and speed by the setting of the pruning threshold.

8.4.2 Sorting

An alternative to the pruning strategy discussed in the previous section is to use fixed beam width. In this case the states are first sorted according to their score $\delta_j(t)$. After that the first N states are expanded and the rest are pruned.

The advantage of this approach is that there is a guaranteed upper bound on the recognition speed since the number of active states never exceeds the given threshold. The main downside to this method is the computational overhead caused by the need to sort the states in the active list. Also the ability of the variable beam width to adjust the beam size according to uncertainty in acoustic model probabilities can mean that beam search may lead to a lower number of active states on average an thus to higher speeds.

Figure 8.7 compares beam search and sorting with a neural network acoustic model. The results show that beam search indeed needs less active states on average to achieve the same accuracy, but the difference is quite small. The last row of graphs shows the relation between recognition speed and accuracy. The additional computation is most likely caused by the sorting algorithm.

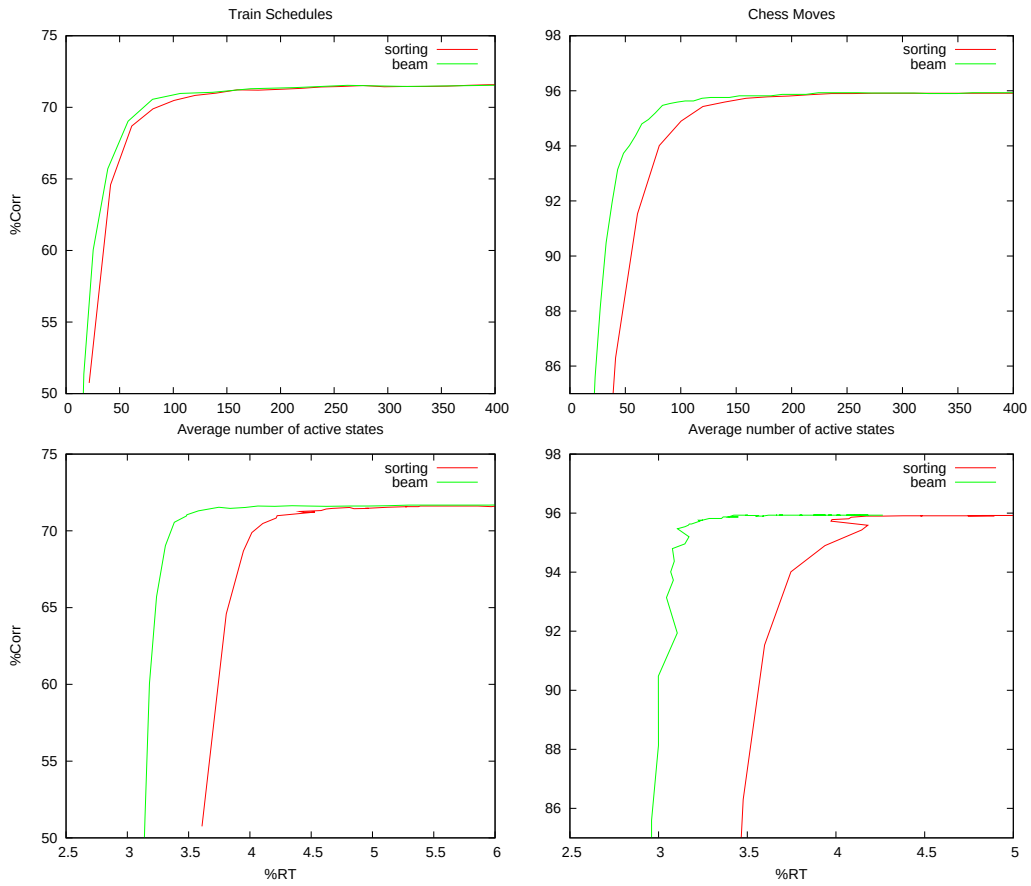

Figure 8.7: The accuracy response to the setting of the beam threshold.

8.4.3 Word Transition Penalty

When observing the output of a recognizer running without a language model (as is the case of our tests with the train schedule corpus) it can be seen that the recognized utterances contain more words than the referential utterances. In many cases there are sequences of recognized short words that sound

similar to the longer correct word. Other common error is an inserted word. Both can be seen in the following example that was found in our test corpus.

This is the correct transcription of the utterance:

`kdy odjíždí vlak do přeštic přibližně ve dvě hodiny odpoledne`

This is the output of the recognizer:

`kdy odjíždí vlak do přeštic s přibližně ve dvě hodiny od pole dne`

In this case the word "s" (with) was inserted and the word "odpoledne" (afternoon) was replaced by the sequence "od" (from) "pole" (field) "dne" (day) which sounds exactly the same. The number of insertion errors can

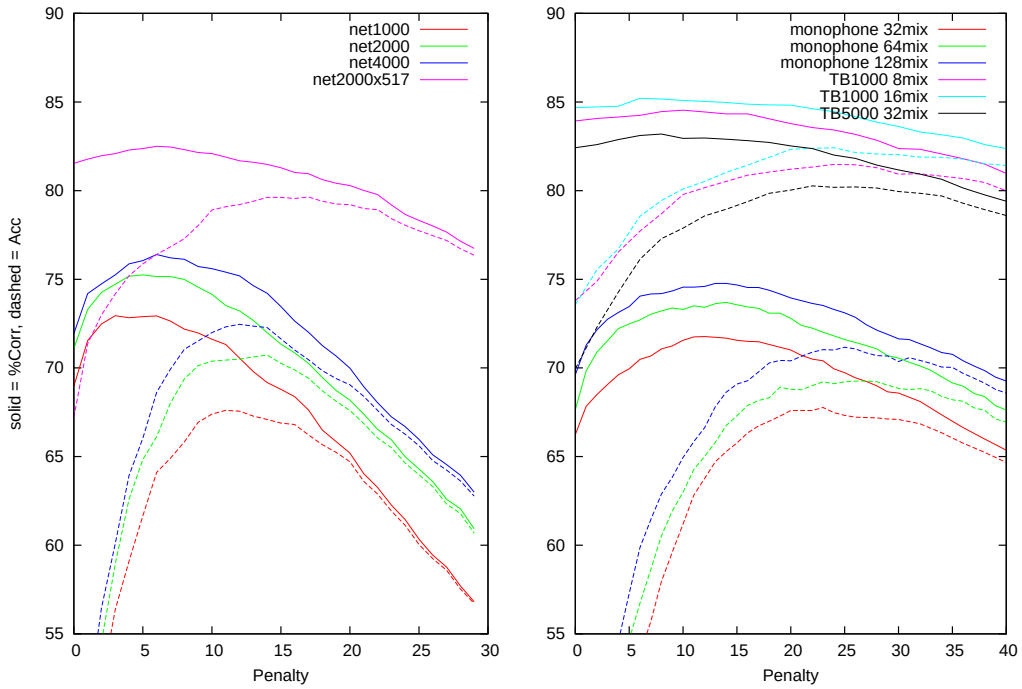

Figure 8.8: The accuracy response to the setting of the word transition penalty. Solid lines represent the measure $\%Corr$, dashed lines represent Acc.

be reduced (and subsequently the value of $\%Acc$ increased) if, during the decoding phase, a penalty ρ is applied to the score when transiting between words:

$$\delta_j(t+1) = \delta_i(t)\rho a_{ij} b_j(o_{t+1}), \qquad \begin{array}{ll} 0 < \rho < 1 & \text{if } i \neq j \text{ and } i \text{ is word end state} \\ \rho = 1 & \text{otherwise} \end{array} \tag{8.4}$$

The penalty is again input in logarithmic form as β:

$$\rho = 10^{-\beta} \tag{8.5}$$

How the recognition system responds to different settings of the word transition penalty can be seen in Figure 8.8. The introduction of the penalty is beneficial for the recognition accuracy regardless of whether it is measured as $\%Corr$ or Acc. With the increasing value of the penalty the measure Acc rises at the expense of $\%Corr$. At this point the prospective user of the final application must decide whether to maximize the total number of correctly recognized words or to minimize insertion errors.

8.5 Language Modeling

In automatic speech recognition there are two independent sources of information that are used to find the most likely word sequence (see equation 4.2): the acoustic model likelihood $P(O \mid W)$ and the prior probability of the word sequence $P(W)$ which is provided by the language model. Because our goal is to compare different kinds of acoustic models we have decided not to use any stochastic language models during the search. This means lower recognition accuracy but the comparison of the acoustic models is still valid, since the acoustic model likelihoods are independent from the language model probabilities.

The choice and tuning of a language model for a given task (the train schedule corpus in our case) is a complicated task and so we have decided that it is beyond the scope of this thesis. However some of our experiments with class based models can be found in [Pav06].

This applies to the trains corpus only. Grammars are available for the other two test corpora (numbers and chess moves). There is no recursion and so the grammars together with pronunciation dictionary can be easily converted to HMM graphs.

Chapter 9

Conclusions

As has been stated in section 4.4 the works of several authors show that neural network acoustic models have advantages over the more commonly used Gaussian mixture models. In this work we have described a variety of experiments that were carried out with the speech corpora we have available in order to maximize our recognition accuracy. Both kinds of acoustic models allow making a choice between computational requirements and recognition accuracy by controlling the total number of trainable parameters. This can be done by choosing the number of mixtures per state in the case of GMMs and the number of hidden neurons in the case of neural network models. In this chapter we will show how the choice of acoustic model and its number of trainable parameters affects the performance of the whole recognition system.

9.1 The Models

To make the final conclusions we have chosen a set of representative models which will be used to demonstrate the differences between Gaussian mixture and neural network models. Altogether there are four general kinds of models:

- **GMM monophones**. There are three denoted `mono32mix`, `mono64mix`, and `mono128mix` based on the number of Gaussians per state.
- **GMM triphones**. The lowest clustering threshold (see Table 6.1) which lead to the highest number of physical states also gave the best results in terms of word recognition accuracy. We have chosen the best performing model `triph16mixTB1000` and its faster version `triph8mixTB1000`. The last triphone model denoted `triph32mixTB5000`

is the one used to generate the training data for the neural network triphone model.

- **Neural network monophones**. These have 117 input units (9 frame context window, see section 7.1.1), 36 output units and vary in the number of hidden units: `net1000`, `net2000` and `net4000`.

- **Neural network triphones**. There is only one network (denoted `net2000x517`) having 2000 hidden units and 517 output neurons representing models at leaves of a decision tree.

Detailed information about these models as well as results for both corpora can be found in appendix C.

9.2 Frame Level Results

The first comparison will be made on frame recognition error (see equation 5.4) and will test how well do the acoustic models perform as classifiers. Before all the models can be compared several adjustments need to be made:

- All the models will be tested on monophone recognition error (otherwise the general trend is that the frame error rate increases with the number of phonetic units). The testing of monophones is straightforward. Triphones need to be converted in the following way: First, the highest scoring triphone for the given frame is chosen. The central monophone of this triphone is then compared with the monophone class label of the frame to find out whether frame error occurred.

- The outputs of the GMM models are likelihoods $P(o|S_j)$ but posteriors $P(S_j|o)$ are better for classification[1]. In order to classify by posteriors the likelihoods are multiplied by class prior probabilities $P(S_j)$ before the frame classification is performed.

The results can be found in Figure 9.1. It can be seen that the frame error rate decreases with the rise of the number of trainable parameters (i.e. the number of hidden neurons or the number of mixtures per state). The other important result of this test is that the neural network models are much better frame level classifiers than their GMM counterparts. As we will show in the following section, this does not always translate to better word level accuracy.

[1]We have tested the classifiers with both likelihood and posterior probabilities and the posteriors always gave better results.

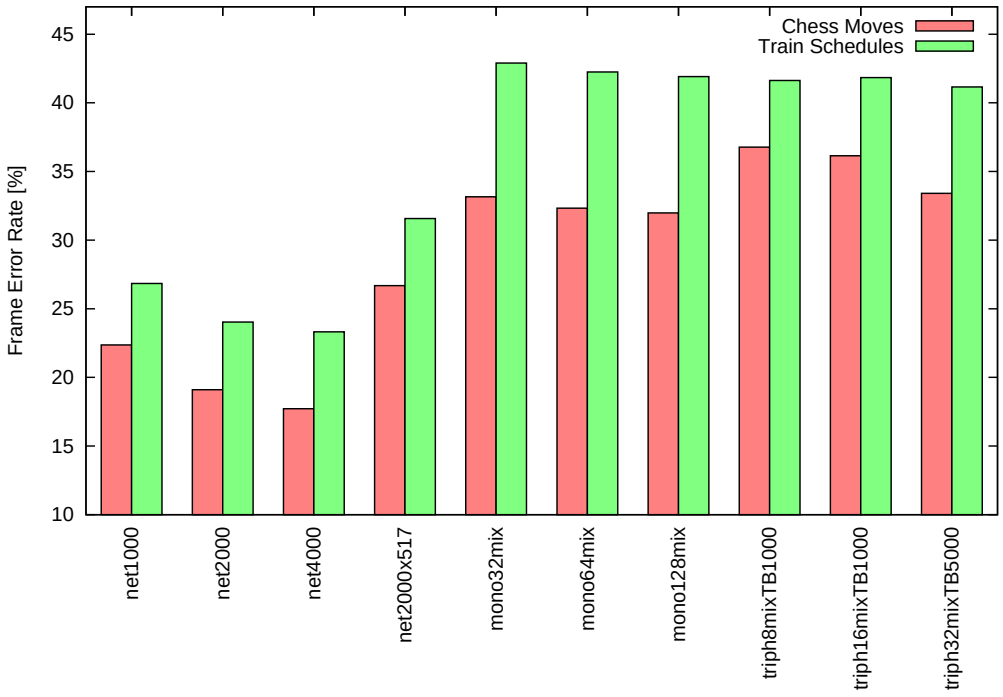

Figure 9.1: Frame error rate for the set of representative models.

9.3 Word Level Results

When evaluating the performance of different acoustic models the most accurate information can be obtained by measuring how well they perform as a part of the whole recognition system. In our opinion the most important measures are the recognition accuracy (in Figure 9.2 represented by $\%Corr$, see appendix C for the values of Acc) and the recognition speed depicted as percentage of real time ($\%RT$, see section 5.1.3).

By adjusting the parameters of the acoustic model (i.e. by changing the number of trainable parameters, changing the threshold for triphone clustering) and by tuning the parameters of the decoder (the beam threshold) one can choose a compromise between recognition accuracy and recognition speed. Figure 9.2 shows where our chosen representative models lie on the speed vs. accuracy plane. It should be noted that while the parameters of the acoustic models are given, the position of each model on the plane can vary according to the chosen beam threshold. Each acoustic model is depicted as a point while a more complete representation would be a curve that represents different values of speed and accuracy based on the beam threshold as it is portrayed in the last row of Figure 8.6.

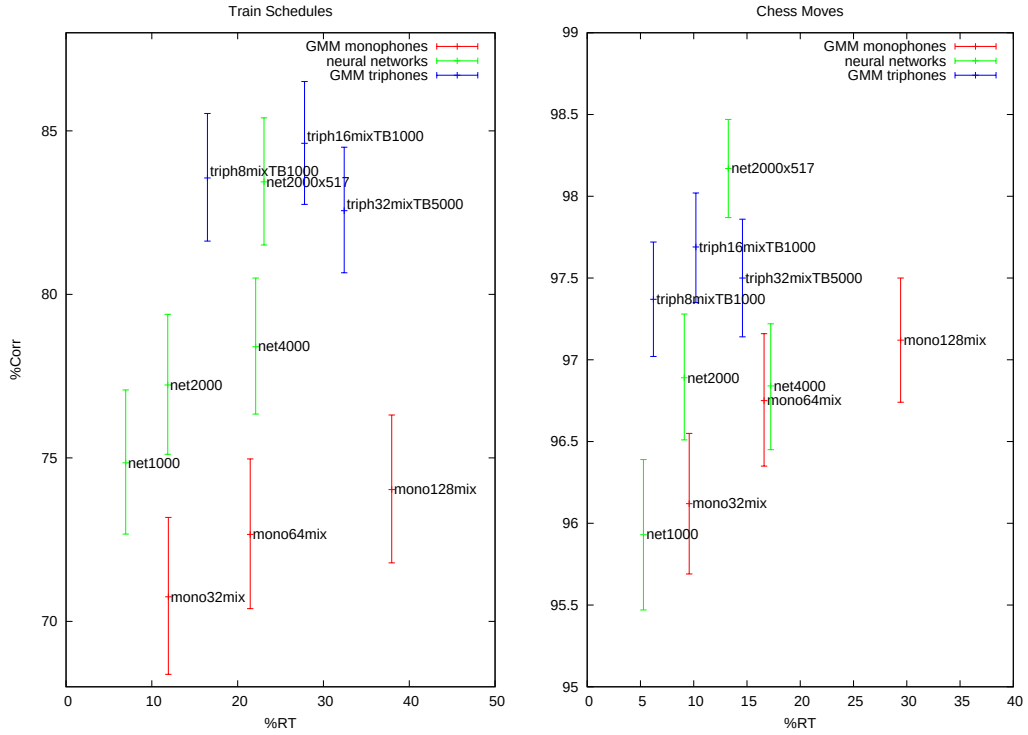

Figure 9.2: The performance of the different acoustic models measured in terms or recognition accuracy and speed. The error bar widths were computed for 99% confidence. For details of computation see appendix B.

The beam thresholds in the experiments with our representative acoustic models were chosen in such a way that the recognition speed with each model would be as high as possible without affecting the relative standing of the models in terms of recognition accuracy. The following observations can be made based on the results of our experiments:

- Triphone models universally outperform monophone models in terms of recognition accuracy (this was expected).
- For monophone models the neural networks give significantly better performance in speed as well as recognition accuracy (especially notable on the train schedule corpus). This is consistent with what the proponents of the hybrid approach say about neural network acoustic models: the neural networks are more economical and require less trainable parameters to obtain the same recognition accuracy as their GMM counterparts.
- The case of triphones is more complicated: If we compare the GMM

model (`triph32mixTB5000`) that was used to generate training labels for our triphone neural network (`net2000x517`) the number of trainable parameters is almost the same (see appendix C). But for the train schedule corpus the neural network gives better speed and for the chess corpus the accuracy is significantly better. The other triphone models with larger number of physical state models show that the GMM based triphones clearly benefit from the conditional computation that is done during pruned decoding run. The larger number of physical state models leads to a higher recognition accuracy but since the search is pruned the computational requirements stay in satisfactory levels. In the case of the train schedule corpus the triphone neural network is outperformed (although the difference may not be statistically significant) in terms of recognition accuracy (by `triph16mixTB1000`) and in terms of recognition speed (by `triph16mixTB1000`).

- For the chess moves corpus the best recognition accuracy was achieved by the triphone neural network. We do not have any indication as to why it happened but it should be noted that the chess moves corpus is very specific one and we have found other occasions where the results obtained on this corpus were unexpected (see e.g. the discussion about prior probabilities in section 7.3).

9.4 Final Notes

Throughout the making of this work we have carried out a large number of experiments with neural network acoustic models as well as with GMM based ones. It should be noted that the results are affected by the training and testing corpora that we have available. The corpora are, by today's standards, quite small and that certainly had an effect on the results that we have obtained. Based on these experiments we present our final conclusions about advantages and disadvantages of neural network based acoustic models.

9.4.1 Advantages of the Hybrid Approach

The advantages of the so called hybrid approach that are proposed by other authors were summarized in section 4.4. According to our experiments we can conclude the following:

- Neural networks are better frame level classifiers. This can be useful in applications that require recognition of phonemes rather than

words. Phoneme recognizers are sometimes used in applications such as keyword spotting or language identification.

- With context independent phonetic units (monophones) neural network models give better results and lead to a higher recognition speed. This can be useful in simple command based tasks when the hardware computational capabilities are limited. Monophones can be useful when working with a domain for which the recognizer was not trained. As Table 7.1 shows the better modeling capabilities of models with more trainable parameters lead to overtraining when viewed from a perspective of using a different testing corpus (the numbers corpus in that case).

- State duplication allows for an easy way of controlling phoneme duration constraints. If, for example, the speaking rate is too fast for three state phoneme models it is trivial to fall back to two state phoneme models if a neural network acoustic model with state duplication is used.

- In a small vocabulary task (such as our chess corpus) the total number of triphones may be sufficiently small that they can be modeled by a neural network. In that case the neural network may still retain the properties that are observed in the monophone case, namely the more economical modeling of the class boundaries, and lead to a better recognition accuracy because of context dependence.

9.4.2 Disadvantages of the Hybrid Approach

- Even though the training software was written in C and was (in our honest opinion) well optimized the time needed for training of a single neural network is extreme when compared to a typical GMM based model. Not only does such a long training hinder a thorough experimentation with the training parameters (such as the learning rate) but also events such as power and network connection (if done in distributed environment) outages must be taken into account.

- In the case of GMMs phonetic unit likelihoods need the whole distribution of the inputs of each class to be modeled. On the other hand if classification is done according to class posteriors it is only necessary to model boundaries between classes in the input vector space. But, boundaries become harder to model with large number of classes if discrimination among classes is required. If GMM models are trained by

maximum likelihood criterion then discrimination among classes is not required and the number of trainable parameters required to model the distribution of class inputs does not increase with the rising number of classes.

This is the case of triphone GMM models: the total number of trainable parameters rises with the increasing number of triphones (or physical states) but the number of parameters (number of mixtures) per triphone (state) can remain constant. If such a model is used by a decoder together with pruning, some state probabilities (in fact a large proportion) need not be computed at all.

As our experiments have shown, using larger number of physical state models increases recognition accuracy for larger vocabulary tasks (such as our train schedule corpus). We suspect that for even larger vocabularies the total number of physical state models would have to increase even further. This brings us to the most important disadvantage of neural network models: GMM likelihood models are much better suited for this task. A neural network would not only require more computation resources for training but it would also require more hidden neurons to properly discriminate among classes.

Bibliography

[Aub02] Aubert, X.L.: An overview of decoding techniques for large vocabulary continuous speech recognition. In: *Computer Speech and Language*. Vol. 16, pp. 89–114, 2002.

[Bat03] Batůšek, R. et al.: Czech SAMPA, `http://noel.feld.cvut.cz/sampa`, 2003.

[Bil98] Bilmes, J.A.: *A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models*, Technical Report, University of Berkeley, ICSI-TR-97-021, 1997.

[BM97] Bourlard, H. and Morgan, N.: Hybrid HMM/ANN Systems for Speech Recognition: Overview and New Research Directions, *Summer School on Neural Networks*, 1997.

[Bou92] Bourlard, H., Morgan, N., Wooters, C., and Renals, S.: CDNN: A Context Dependent Neural Network for Continuous Speech Recognition, *Acoustics, Speech, and Signal Processing ICASSP-92*, 1992, vol. 2, pp. 349 – 352.

[Coh92] Cohen, M. et al.: Hybrid Neural Network/Hidden Markov Model Continuous Speech Recognition, *Proc. of the International Conference on Spoken Language Processing*, 1992.

[Cyb89] Cybenko, G.: Approximation by Superpositions of a Sigmoidal Function, *Math. Control Signals Systems*, 2, 303-314, 1989.

[EP04a] Ekštein, K., Pavelka, T.: LINGVO/LASER: Prototyping Concept of Dialogue Information System with Spreading Knowledge, *Proc. of NLUCS 2004*, Porto, Portugal, 2004.

[EP04b] Ekštein, K., Pavelka, T.: Entropy and Entropy-based Features in Signal Processing. *Proceedings of PhD Workshop Systems & Control*, Balatonfüred 2004.

[Hab08] Habernal, I., Konopík, M.: Active Tags for Semantic Analysis, *Proceedings of TSD2008*, Brno, Czech Republic, 2008.

[Has95] Hassoun, M.H.: *Fundamentals of Artificial Neural Networks*, MIT Press, 1995.

[Hej07] Hejtmánek, J.: *Využití kontextově závislých fonetických jednotek při rozpoznávání přirozené řeči*, M.S. thesis, University of West Bohemia, Plzeň, Czech Republic, June 2007.

[Her90] Hermansky, H.: Perceptual Linear Predictive (PLP) Analysis for Speech, *Journal of Acoustic Society of America*, 1990.

[HM91] Hermansky, H., Morgan, N., Bayya, A., Kohn, P.: *RASTA-PLP Speech Analysis*, Technical Report, ICSI-TR-91-069, 1991.

[Hos00] Hossom, J.P.: *Automatic Time Alignment of Phonemes Using Acoustic-Phonetic Information*, PhD Thesis, Oregon Graduate Institute of Science and Technology, 2000.

[HL87] Huang, W.Y. and Lippmann, R.P.: Neural Nets and Traditional Classifiers, *Neural Information Processing Systems*, 387-396, 1987.

[Hu96] Hu, Z. et al.: Speech Recognition Using Syllable-Like Units, *Proc. of ICSLP*, 1996.

[Hua92] Huang, X. et al.: *The SPHINX-II Speech Recognition System: An Overview*, Technical Report, CMU-CS-92-112, Carnegie Mellon University, 1992.

[Hun99] Hunt, M.J.: Spectral Signal Processing for ASR, *Workshop on Automatic Speech Recognition and Understanding, ASRU-99*, 1999.

[IPA99] International Phonetic Association (Corporate Author): *Handbook of the International Phonetic Association: A guide to the use of the International Phonetic Alphabet*, Cambridge University Press, 1999.

[JM00] Jurafsky, D. and Martin, J.: *Speech and Language Processing*, Prentice Hall, 2000.

[KS96] Kröse, B., van der Smagt, P.: *An Introduction to Neural Networks*, The University of Amsterdam, 1996, can be found at `http://www.robotic.dlr.de/Smagt/books/neuro-intro.ps.gz`.

[Lev66] Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady 10* (Engish translation), 707–710, 1966.

[MP43] McCulloch, W.S. and Pitts, W.: A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 115–133, 1943.

[Nis07] *NIST/SEMATECH e-Handbook of Statistical Methods*, can be found online at `http://www.itl.nist.gov/div898/handbook/`, 2007.

[Ode95] Odell, J.J.: *The Use of Context in Large Vocabulary Speech Recognition*, PhD Thesis, Cambridge University Engineering Dept, 1995.

[Pav03] Pavelka, T.: *Hybrid Speech Recognizer Implementation*, Diploma Thesis, University of West Bohemia, 2003.

[Pav06] Pavelka, T.: LDec: One Pass Time Synchronous Decoder, *Proc. of PhD Workshop 2006*, Hrubá Skála. Czech Republic, 2006

[Pav07] Pavelka, T., Ekštein, K.: JLASER: An Automatic Speech Recognizer Written in Java, *Proc. of XII International Conference Speech and Computer (SPECOM'2007)*, Moscow, Russia, 2007.

[PM88] Peeling, S.M. and Moore, R.K.: Isolated Digit Recognition Experiments Using the Multi-layer Perceptron, *Speech Communication 7*, 1988.

[Rab89] Rabiner, L.R.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, *Proceedings of the IEEE*, vol. 77, no. 2, 1989

[Rie93] Riedmiller, M. and Braun, H.: A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm, *Proceedings of the IEEE International Conference on Neural Networks*, 1993.

[RH95] Rennals, S., Hochberg, M.: Efficient search using posterior phone probability estimates, *Proc. of IEEE ICASSP*, 1995.

[RH97] Rennals, S., Hochberg, M.: Start-Synchronous Search for Large Vocabulary Continuous Speech Recognition, *Proc. of IEEE Transactions on Speech and Audio Processing*, 1997.

[Rob89] Robinson, A.J.: *Dynamic Error Propagation Networks*, PhD Thesis, Cambridge University Engineering Dept., 1989

[Rob94] Robinson, A.J.: An Application of Recurrent Nets to Phone Probability Estimation, *IEEE transactions on neural networks*, Volume 5, Number 3, 1994.

[Rob02] Robinson, A.J. et al.: Connectionist Speech Recognition of Broadcast News, *Speech Communication*, 37, 27-45, 2002.

[Rum86] Rumelhart, D., Hinton, G., Williams, R.: Learning Internal Representations by Error Propagation, *Parallel Distributed Processing* Vol.1, 318-362, Cambridge, MA, MIT Press, 1986.

[Sar02] Sarle, W.S.: *Neural Network FAQ*, periodic posting to the Usenet newsgroup comp.ai.neural-nets, ed. 2002,
URL: `ftp://ftp.sas.com/pub/neural/FAQ.html`

[Ser97] Serridge, B.M.: *Context-Dependent Modeling in a Segment-Based Speech Recognition System*, PhD Thesis, Dept. of Electrical Engineering and Computer Science, MIT, 1997.

[Sch93] Schukat-Talamazzini, E.G., Niemann, H., Eckert, W., Kuhn, T., Rieck S.: Automatic Speech Recognition without Phonemes, *In Proc. European Conf. on Speech Communication and Technology*, 1993.

[Sut98] Sutton, S. et al.: Universal Speech Tools: the CSLU Toolkit, *Proc. of the ICSLP*, 1998.

[Teb95] Tebelskis, J.: *Speech Recognition using Neural Networks*, PhD Thesis, Carnegie Mellon University, 1995.

[Vav08] Vávra, F., Pavelka, T., Šedivá, B., Vokáčová, K., Marek, P., Neumanová, M.: Ratio Statistics, *JČMF ROBUST 2008*, Pribylina, Slovensko, 2008.

[Wal04] Walker, W. et al.: *Sphinx-4: A Flexible Open Source Framework for Speech Recognition*, technical report SMLI TR-2004-139, November 2004.

[You89] Young, S.J., Russell, N.H., Thornton, J.H.S.: *Token Passing: A Simple Conceptual Model for Connected Speech Recognition Systems*, Technical Report CUED/F-INFENG/TR38, Cambridge University Engineering Dept, 1989.

[You02] Young, S. et al., *The HTK Book (for HTK v. 3.1)*, Cambridge University Engineering Dept, 2002.

Author's Publications

The following papers were published in conference proceedings:

1. Ekštein, K., Matoušek, V., Pavelka, T.: Automatic Segmentation and Labeling of Speech Signal, *Proceedings of Elektronische Sprachsignalverarbeitung*, Universitätsverlag & Buchhandel, Dresden, Germany, 2003.

2. Brey, T. and Pavelka, T.: A Speech Platform for a Bilingual City Information System, *Proceedings of TSD 2004*, Brno, Czech Republic, 2004.

3. Ekštein, K., Pavelka, T.: LINGVO/LASER: Prototyping Concept of Dialogue Information System with Spreading Knowledge, *Proceedings of the 1st International Workshop on Natural Language Understanding and Cognitive Sciences, NLUCS 2004*, INSTICC Press, Porto, Portugal, 2004.

4. Ekštein, K. and Pavelka T.: Entropy And Entropy-based Features In Signal Processing *Proceedings of PhD Workshop Systems & Control*, Balatonfüred, Hungary, 2004.

5. Pavelka, T. and Ekštein K.: Low Level Performance on Continuous Speech: Humans vs. Computers, *Proceedings of PhD Workshop Systems & Control*, Balatonfüred, Hungary, 2004.

6. Pavelka, T., Ekštein, K., Andrš, D.: Hybridní rozpoznávač přirozené řeči pro český jazyk, *Proceedings of Kognícia a umelý život 2005*, Smolenice, Slovakia, 2005

7. Pavelka, T., Ekštein, K.: Neural Network Acoustic Model for Recognition of Czech Speech, *proc. Proceedings of PhD Workshop Systems & Control*, Izola, Slovenia, 2005

8. Král, P., Cerisara, C., Klečková, J., Pavelka, T.: Sentence Structure for Dialog Act Recognition in Czech, *Proceedings of ICTTA ´06*, Damascus, Syria, 2006.

9. Pavelka, T.: LDec: One Pass Time Synchronous Decoder, *Proc. of PhD Workshop 2006*, Hrubá Skála. Czech Republic, 2006.

10. Pavelka, T. and Hejtmánek, J.: Context Dependency in Neural Network Based Acoustic Models, *Proceedings of PhD Workshop Systems & Control*, Balatonfüred, Hungary, 2007.

11. Hejtmánek, J. and Pavelka, T.: Use of Context-Dependent Units in Czech Speech Recognition, *Proceedings of PhD Workshop Systems & Control*, Balatonfüred, Hungary, 2007.

12. Pavelka, T. and Ekštein, K.: JLASER: An Automatic Speech Recognizer Written in Java, *Proc. of XII International Conference Speech and Computer (SPECOM'2007)*, Moscow, Russia, 2007.

13. Pavelka, T., Král, P.: Neural Network Acoustic Model with Decision Tree Clustered Triphones, *Proceedings of 2008 IEEE International Workshop on Machine Learning for Signal Processing*, Cancún, Mexico, 2008.

14. Král, P., Pavelka, T.: Evaluation of Dialogue Act Recognition Approaches, *Proceedings of 2008 IEEE International Workshop on Machine Learning for Signal Processing*, Cancún, Mexico, 2008.

15. Pavelka, T., Brychcín, T.: N-Best Decoder for the JLASER Automatic Speech Recognizer, *Proceedings of 9th International PhD Workshop on Systems and Control (YGV2008)*, Izola, Slovenia, 2008.

16. Hejtmánek, J., Pavelka, T.: Automatic Speech Recognition Using Context-dependent Syllables, *Proceedings of 9th International PhD Workshop on Systems and Control (YGV2008)*, Izola, Slovenia, 2008.

17. Vávra, F., Pavelka, T., Šedivá, B., Vokáčová, K., Marek, P., Neumanová, M.: Ratio Statistics, *Proceedings of JČMF ROBUST 2008*, Pribylina, Slovakia, 2008.

The following papers were published as technical reports at the Department of Computer Science and Engineering, University of West Bohemia.

1. Pavelka, T.: *Hybrid Methods of Automatic Speech Recognition, PhD thesis exposé*, technical report No. DCSE/TR-2005-07, University of West Bohemia, Pilsen, 2005.

Appendix A

LASER Phonetic Alphabet

Vowels

CLS symbol	Orthographic	Phonetic
a	pas	p a s
e	les	l e s
i	pivo	p i v o
o	rok	r o k
u	rum	r u m
a_	řád	r a_ t
e_	lék	l e_ k
i_	pít	p i_ t
o_	móda	m o_ d a
u_	půl	p u_ l

Plosives

CLS symbol	Orthographic	Phonetic
p	pes	p e s
t	tam	t a m
t’	tito	t’ i t o
k	kam	k a m
b	bota	b o t a
d	dům	d u_ m
d’	děd	d’ e t
g	kde	g d e

Affricates

CLS symbol	Orthographic	Phonetic
ts	cíl	ts i_ l
ts’	čas	ts’ a s

Fricatives

CLS symbol	Orthographic	Phonetic
f	forma	f o r m a
s	sen	s e n
r’	řeka	r’ e k a
s’	šála	s’ a_ l a
x	chlapec	x l a p e ts
v	vak	v a k
z	zub	z u p
z’	žal	z’ a l
j	boj	b o j
h	had	h a t

Liquids

CLS symbol	Orthographic	Phonetic
r	ret	r e t
l	led	l e t

Nasals

CLS symbol	Orthographic	Phonetic
m	mrak	m r a k
n	noc	n o ts
n’	nic	n’ i ts

Appendix B

Confidence Interval Computation

Let $(r_1, s_1), (r_2, s_2), \ldots, (r_n, s_n)$ be a set of measurements of sentence recognition accuracy where r_i is the number of correctly recognized words in sentence i (i.e. $r_i = H_i$ in notation used in eq. 5.1), and s_i is the number of incorrectly recognized words (i.e. $s_i = N_i - H_i$) Let ξ_n be a random variable representing a ratio

$$\xi_n = \frac{\sum_{i=1}^{n} r_i}{\sum_{i=1}^{n} r_i + \sum_{i=1}^{n} s_i} = \frac{H}{N}. \tag{B.1}$$

Suppose that the expected value estimations

$$e_r = \frac{1}{n}\sum_{i=1}^{n} r_i, \quad e_s = \frac{1}{n}\sum_{i=1}^{n} s_i \tag{B.2}$$

are known as well as the variance estimations

$$\sigma_r^2 = \frac{1}{n}\sum_{i=1}^{n}(r_i - e_r)^2, \quad \sigma_s^2 = \frac{1}{n}\sum_{i=1}^{n}(s_i - e_r)^2 \tag{B.3}$$

and the correlation coefficient

$$\rho_{r,s}\sigma_r\sigma_s = \frac{1}{n}\sum_{i=1}^{n}(r_i - e_r)(s_i - e_s). \tag{B.4}$$

Because we have sufficiently large number of samples the estimates do not significantly differ from the abstract expected values. The distribution function of the random variable ξ_n (see [Vav08] to see how it is inferred) can be

expressed as

$$F_{\xi_n} = 1-\Phi\left(-\sqrt{n}\frac{e_s-\left(\frac{1}{x}-1\right)e_r}{\sqrt{\sigma_s^2+\left(\frac{1}{x}-1\right)^2\sigma_r^2-2\left(\frac{1}{x}-1\right)\rho_{r,s}\sigma_r\sigma_s}}\right) \Leftrightarrow 0<x<1. \tag{B.5}$$

We are looking for a confidence interval $(x_L; x_U)$ that satisfies the condition

$$P(x_L \leq \xi_n \leq x_U) = 1-\alpha. \tag{B.6}$$

To uniquely determine the bounds we chose α_1 and α_2 so that $\alpha = \alpha_1 + \alpha_2$; $0 < \alpha, \alpha_1, \alpha_2 < 1$. The interval boundaries can then be found by solving the equations

$$F_{\xi_n}(x_U) = 1-\alpha_1 \tag{B.7}$$

and

$$F_{\xi_n}(x_L) = \alpha_2. \tag{B.8}$$

B.1 Determination of the Lower Bound

The value of x_L is the solution to the equation

$$1-\Phi\left(-\sqrt{n}\frac{e_s-\left(\frac{1}{x_L}-1\right)e_r}{\sqrt{\sigma_s^2+\left(\frac{1}{x_L}-1\right)^2\sigma_r^2-2\left(\frac{1}{x_L}-1\right)\rho_{r,s}\sigma_r\sigma_s}}\right) = \alpha_2. \tag{B.9}$$

From this we get

$$\Phi^{-1}(1-\alpha_2) = -\sqrt{n}\left(\frac{e_s-\left(\frac{1}{x_L}-1\right)e_r}{\sqrt{\sigma_s^2+\left(\frac{1}{x_L}-1\right)^2\sigma_r^2-2\left(\frac{1}{x_L}-1\right)\rho_{r,s}\sigma_r\sigma_s}}\right). \tag{B.10}$$

By substituting $z = \frac{1}{x_L} - 1$ we get

$$\Phi^{-1}(1-\alpha_2) = -\sqrt{n}\frac{e_s - ze_r}{\sqrt{\sigma_s^2+z^2\sigma_r^2-2z\rho_{r,s}\sigma_r\sigma_s}}, \tag{B.11}$$

from this

$$\Phi^{-1}(1-\alpha_2)\sqrt{\sigma_s^2+z^2\sigma_r^2-2z\rho_{r,s}\sigma_r\sigma_s}=-\sqrt{n}(e_s-ze_r). \tag{B.12}$$

Next, the equation is squared:

$$\left[\Phi^{-1}(1-\alpha_2)\right]^2(\sigma_s^2+z^2\sigma_r^2-2z\rho_{r,s}\sigma_r\sigma_s)=n(e_s-ze_r). \tag{B.13}$$

To transform the equation into a more readable form we will use the substitutuion

$$Az^2+2zB+C=0 \tag{B.14}$$

where $A=(a\sigma_r^2-e_r^2)$; $\quad B=(e_re_s-\rho_{r,s}a\sigma_r\sigma_s)$; $\quad C=(a\sigma_s^2-e_s^2)$, and $a=\dfrac{\left[\Phi^{-1}(1-\alpha_2)\right]^2}{n}$. By applying the common solution to the quadratic equation we get

$$z_{1,2}=\frac{-B\pm\sqrt{B^2-AC}}{A} \tag{B.15}$$

Only one of the roots is valid and the validity is checked by the equation

$$\frac{\Phi^{-1}(1-\alpha_2)}{\sqrt{n}}\sqrt{\sigma_s^2+z^2\sigma_r^2-2z\rho_{r,s}\sigma_r\sigma_s}+(e_s-ze_r)=0. \tag{B.16}$$

Once the correct root is found, the lower bound can be computed:

$$x_L=\frac{1}{1+z}. \tag{B.17}$$

B.2 Determination of the Upper Bound

The value of x_U is the solution to the equation

$$\Phi\left(-\sqrt{n}\frac{e_s-\left(\frac{1}{x_U}-1\right)e_r}{\sqrt{\sigma_s^2+\left(\frac{1}{x_U}-1\right)^2\sigma_r^2-2\left(\frac{1}{x_U}-1\right)\rho_{r,s}\sigma_r\sigma_s}}\right)=\alpha_1. \tag{B.18}$$

The way of finding the value of the upper bound x_U is very similar to the calculation of x_L described above except that

$$a=\frac{\left[\Phi^{-1}(\alpha_1)\right]^2}{n} \tag{B.19}$$

and the root validation equation is

$$\frac{\Phi^{-1}(\alpha_1)}{\sqrt{n}}\sqrt{\sigma_s^2+z^2\sigma_r^2-2z\rho_{r,s}\sigma_r\sigma_s}+(e_s-ze_r)=0. \tag{B.20}$$

Appendix C

Detailed Results

All confidence intervals were computed by the method described in appendix B for 99% probability, i.e. $\alpha_1 = 0.005$ and $\alpha_2 = 0.005$.

C.1 Train Schedules

C.1.1 net1000

- **Acoustic model**
  - Type: neural network
  - Number of unique state models: 36
  - Input layer: 117 neurons, identity activation function
  - Hidden layer: 1000 neurons, symmetric logistic activation function
  - Output layer: 36 neurons, logistic activation function
  - Number of trainable parameters: 155000
- **Decoder**
  - beam: 10^{12}
  - penalty: 10.0
- **Results**
  - % of correct sentences: 11.90
  - $\%Corr$: 74.75, confidence interval: (72.59;76.96)
  - Acc: 70.07, confidence interval: (66.25;73.90)

- $H = 3811$ $D = 321$ $S = 966$ $I = 239$ $N = 5098$
- %RT: 7.34
- Frame error rate: 26.84%

C.1.2 net2000

- **Acoustic model**
  - Type: neural network
  - Number of unique state models: 36
  - Input layer: 117 neurons, identity activation function
  - Hidden layer: 2000 neurons, symmetric logistic activation function
  - Output layer: 36 neurons, logistic activation function
  - Number of trainable parameters: 310000
- **Decoder**
  - beam: 10^{13}
  - penalty: 10.0
- **Results**
  - % of correct sentences: 12.91
  - $\%Corr$: 77.19, confidence interval: (75.07;79.35)
  - Acc: 73.17, confidence interval: (70.58;75.77)
  - $H = 3935$ $D = 309$ $S = 854$ $I = 205$ $N = 5098$
  - %RT: 10.91
  - Frame error rate: 24.04%

C.1.3 net4000

- **Acoustic model**
  - Type: neural network
  - Number of unique state models: 36
  - Input layer: 117 neurons, identity activation function
  - Hidden layer: 4000 neurons, symmetric logistic activation function

- Output layer: 36 neurons, logistic activation function
- Number of trainable parameters: 620000

- **Decoder**
  - beam: 10^{14}
  - penalty: 10.0

- **Results**
  - % of correct sentences: 15.44
  - %*Corr*: 78.40, confidence interval: (76.35;80.49)
  - *Acc*: 74.83, confidence interval: (72.31;77.38)
  - $H = 3997$ $D = 295$ $S = 806$ $I = 182$ $N = 5098$
  - %RT: 17.98
  - Frame error rate: 23.32%

C.1.4 net2000x517

- **Acoustic model**
  - Type: neural network
  - Number of unique state models: 517
  - Input layer: 117 neurons, identity activation function
  - Hidden layer: 2000 neurons, symmetric logistic activation function
  - Output layer: 517 neurons, logistic activation function
  - Number of trainable parameters: 1272000

- **Decoder**
  - beam: 10^{16}
  - penalty: 10.0

- **Results**
  - % of correct sentences: 24.56
  - %*Corr*: 83.25, confidence interval: (81.29;85.23)
  - *Acc*: 80.66, confidence interval: (78.31;83.01)
  - $H = 4244$ $D = 249$ $S = 605$ $I = 132$ $N = 5098$
  - %RT: 17.63
  - Frame error rate: 31.57%

C.1.5 mono32mix

- **Acoustic model**
  - Type: monophone GMM
  - Number of unique state models: 108
  - Number of mixtures: 32
  - Number of trainable parameters: 269568
- **Decoder**
  - beam: 10^{26}
  - penalty: 20.0
- **Results**
  - % of correct sentences: 9.62
  - $\%Corr$: 70.73, confidence interval: (68.36;73.16)
  - Acc: 67.34, confidence interval: (64.72;70.00)
  - $H = 3606$ $D = 373$ $S = 1119$ $I = 173$ $N = 5098$
  - %RT: 14.52
  - Frame error rate: 42.90%

C.1.6 mono64mix

- **Acoustic model**
  - Type: monophone GMM
  - Number of unique state models: 108
  - Number of mixtures: 64
  - Number of trainable parameters: 539136
- **Decoder**
  - beam: 10^{26}
  - penalty: 20.0
- **Results**
  - % of correct sentences: 9.87

- %$Corr$: 72.81, confidence interval: (70.57;75.10)
- Acc: 68.63, confidence interval: (65.40;71.90)
- $H = 3712$ $D = 341$ $S = 1045$ $I = 213$ $N = 5098$
- %RT: 23.87
- Frame error rate: 42.25%

C.1.7 mono128mix

- **Acoustic model**
  - Type: monophone GMM
  - Number of unique state models: 108
  - Number of mixtures: 128
  - Number of trainable parameters: 1078272
- **Decoder**
  - beam: 10^{26}
  - penalty: 20.0
- **Results**
  - % of correct sentences: 12.41
  - %$Corr$: 74.01, confidence interval: (71.76;76.30)
  - Acc: 70.18, confidence interval: (67.02;73.37)
  - $H = 3773$ $D = 337$ $S = 988$ $I = 195$ $N = 5098$
  - %RT: 40.59
  - Frame error rate: 41.91%

C.1.8 triph8mixTB1000

- **Acoustic model**
  - Type: triphone GMM
  - Number of unique state models: 1773
  - Number of mixtures: 8
  - Number of trainable parameters: 1106352

- **Decoder**
  - beam: 10^{34}
  - penalty: 20.0
- **Results**
  - % of correct sentences: 22.28
  - $\%Corr$: 83.56, confidence interval: (81.63;85.53)
  - Acc: 80.99, confidence interval: (78.81;83.20)
  - $H = 4260$ $D = 295$ $S = 543$ $I = 131$ $N = 5098$
  - %RT: 12.45
  - Frame error rate: 41.63%

C.1.9 triph16mixTB1000

- **Acoustic model**
  - Type: triphone GMM
  - Number of unique state models: 1773
  - Number of mixtures: 16
  - Number of trainable parameters: 2212704
- **Decoder**
  - beam: 10^{34}
  - penalty: 20.0
- **Results**
  - % of correct sentences: 24.05
  - $\%Corr$: 84.62, confidence interval: (82.75;86.51)
  - Acc: 82.13, confidence interval: (79.97;84.30)
  - $H = 4314$ $D = 266$ $S = 518$ $I = 127$ $N = 5098$
  - %RT: 25.39
  - Frame error rate: 41.84%

C.1.10 triph32mixTB5000

- **Acoustic model**
  - Type: triphone GMM
  - Number of unique state models: 517
  - Number of mixtures: 32
  - Number of trainable parameters: 1290432
- **Decoder**
  - beam: 10^{33}
  - penalty: 20.0
- **Results**
  - % of correct sentences: 20.51
  - $\%Corr$: 82.52, confidence interval: (80.61;84.47)
  - Acc: 80.03, confidence interval: (77.80;82.29)
  - $H = 4207 \;\; D = 302 \;\; S = 589 \;\; I = 127 \;\; N = 5098$
  - %RT: 25.94
  - Frame error rate: 41.16%

C.2 Chess Moves

C.2.1 net1000

- **Acoustic model**
  - Type: neural network
  - Number of unique state models: 36
  - Input layer: 117 neurons, identity activation function
  - Hidden layer: 1000 neurons, symmetric logistic activation function
  - Output layer: 36 neurons, logistic activation function
  - Number of trainable parameters: 155000
- **Decoder**
  - beam: 10^{33}

- penalty: 0.0

- **Results**
  - % of correct sentences: 74.39
  - $\%Corr$: 95.93, confidence interval: (95.47;96.39)
  - Acc: 95.92, confidence interval: (95.46;96.38)
  - $H = 12973 \;\; D = 6 \;\; S = 545 \;\; I = 1 \;\; N = 13524$
  - %RT: 4.36
  - Frame error rate: 22.36%

C.2.2 net2000

- **Acoustic model**
  - Type: neural network
  - Number of unique state models: 36
  - Input layer: 117 neurons, identity activation function
  - Hidden layer: 2000 neurons, symmetric logistic activation function
  - Output layer: 36 neurons, logistic activation function
  - Number of trainable parameters: 310000

- **Decoder**
  - beam: 10^{34}
  - penalty: 0.0

- **Results**
  - % of correct sentences: 78.56
  - $\%Corr$: 96.89, confidence interval: (96.51;97.28)
  - Acc: 96.86, confidence interval: (96.47;97.24)
  - $H = 13104 \;\; D = 3 \;\; S = 417 \;\; I = 5 \;\; N = 13524$
  - %RT: 7.37
  - Frame error rate: 19.10%

C.2.3 net4000

- **Acoustic model**
  - Type: neural network
  - Number of unique state models: 36
  - Input layer: 117 neurons, identity activation function
  - Hidden layer: 4000 neurons, symmetric logistic activation function
  - Output layer: 36 neurons, logistic activation function
  - Number of trainable parameters: 620000
- **Decoder**
  - beam: 10^{35}
  - penalty: 0.0
- **Results**
  - % of correct sentences: 78.72
  - $\%Corr$: 96.84, confidence interval: (96.45;97.22)
  - Acc: 96.83, confidence interval: (96.44;97.22)
  - $H = 13096 \;\; D = 3 \;\; S = 425 \;\; I = 1 \;\; N = 13524$
  - %RT: 13.36
  - Frame error rate: 17.72%

C.2.4 net2000x517

- **Acoustic model**
  - Type: neural network
  - Number of unique state models: 517
  - Input layer: 117 neurons, identity activation function
  - Hidden layer: 2000 neurons, symmetric logistic activation function
  - Output layer: 517 neurons, logistic activation function
  - Number of trainable parameters: 1272000
- **Decoder**
  - beam: 10^{45}

  - penalty: 0.0

- **Results**
  - % of correct sentences: 86.94
  - $\%Corr$: 98.17, confidence interval: (97.87;98.47)
  - Acc: 98.17, confidence interval: (97.87;98.47)
  - $H = 13276 \;\; D = 3 \;\; S = 245 \;\; I = 0 \;\; N = 13524$
  - %RT: 9.65
  - Frame error rate: 26.69%

C.2.5 mono32mix

- **Acoustic model**
  - Type: monophone GMM
  - Number of unique state models: 108
  - Number of mixtures: 32
  - Number of trainable parameters: 269568

- **Decoder**
  - beam: 10^{73}
  - penalty: 0.0

- **Results**
  - % of correct sentences: 74.50
  - $\%Corr$: 96.12, confidence interval: (95.69;96.55)
  - Acc: 96.12, confidence interval: (95.69;96.55)
  - $H = 12999 \;\; D = 5 \;\; S = 520 \;\; I = 0 \;\; N = 13524$
  - %RT: 8.34
  - Frame error rate: 33.16%

C.2.6 mono64mix

- **Acoustic model**
  - Type: monophone GMM
  - Number of unique state models: 108
  - Number of mixtures: 64
  - Number of trainable parameters: 539136
- **Decoder**
  - beam: 10^{74}
  - penalty: 0.0
- **Results**
  - % of correct sentences: 78.56
  - $\%Corr$: 96.75, confidence interval: (96.35;97.16)
  - Acc: 96.75, confidence interval: (96.35;97.16)
  - $H = 13085 \;\; D = 2 \;\; S = 437 \;\; I = 0 \;\; N = 13524$
  - %RT: 14.48
  - Frame error rate: 32.33%

C.2.7 mono128mix

- **Acoustic model**
  - Type: monophone GMM
  - Number of unique state models: 108
  - Number of mixtures: 128
  - Number of trainable parameters: 1078272
- **Decoder**
  - beam: 10^{75}
  - penalty: 0.0
- **Results**
  - % of correct sentences: 80.72

- %$Corr$: 97.12, confidence interval: (96.74;97.50)
- Acc: 97.12, confidence interval: (96.74;97.50)
- $H = 13134$ $D = 3$ $S = 387$ $I = 0$ $N = 13524$
- %RT: 26.15
- Frame error rate: 31.99%

C.2.8 triph8mixTB1000

- **Acoustic model**
  - Type: triphone GMM
  - Number of unique state models: 1773
  - Number of mixtures: 8
  - Number of trainable parameters: 1106352
- **Decoder**
  - beam: 10^{110}
  - penalty: 0.0
- **Results**
  - % of correct sentences: 79.89
  - %$Corr$: 97.37, confidence interval: (97.02;97.72)
  - Acc: 97.12, confidence interval: (96.76;97.49)
  - $H = 13168$ $D = 3$ $S = 353$ $I = 33$ $N = 13524$
  - %RT: 5.14
  - Frame error rate: 36.77%

C.2.9 triph16mixTB1000

- **Acoustic model**
  - Type: triphone GMM
  - Number of unique state models: 1773
  - Number of mixtures: 16
  - Number of trainable parameters: 2212704

- **Decoder**
  - beam: 10^{110}
  - penalty: 0.0
- **Results**
  - % of correct sentences: 82.06
  - %$Corr$: 97.69, confidence interval: (97.35;98.02)
  - Acc: 97.40, confidence interval: (97.05;97.76)
  - $H = 13211$ $D = 3$ $S = 310$ $I = 38$ $N = 13524$
  - %RT: 8.58
  - Frame error rate: 36.14%

C.2.10 triph32mixTB5000

- **Acoustic model**
  - Type: triphone GMM
  - Number of unique state models: 517
  - Number of mixtures: 32
  - Number of trainable parameters: 1290432
- **Decoder**
  - beam: 10^{95}
  - penalty: 0.0
- **Results**
  - % of correct sentences: 81.67
  - %$Corr$: 97.50, confidence interval: (97.14;97.86)
  - Acc: 97.20, confidence interval: (96.81;97.58)
  - $H = 13186$ $D = 6$ $S = 332$ $I = 41$ $N = 13524$
  - %RT: 12.60
  - Frame error rate: 33.41%